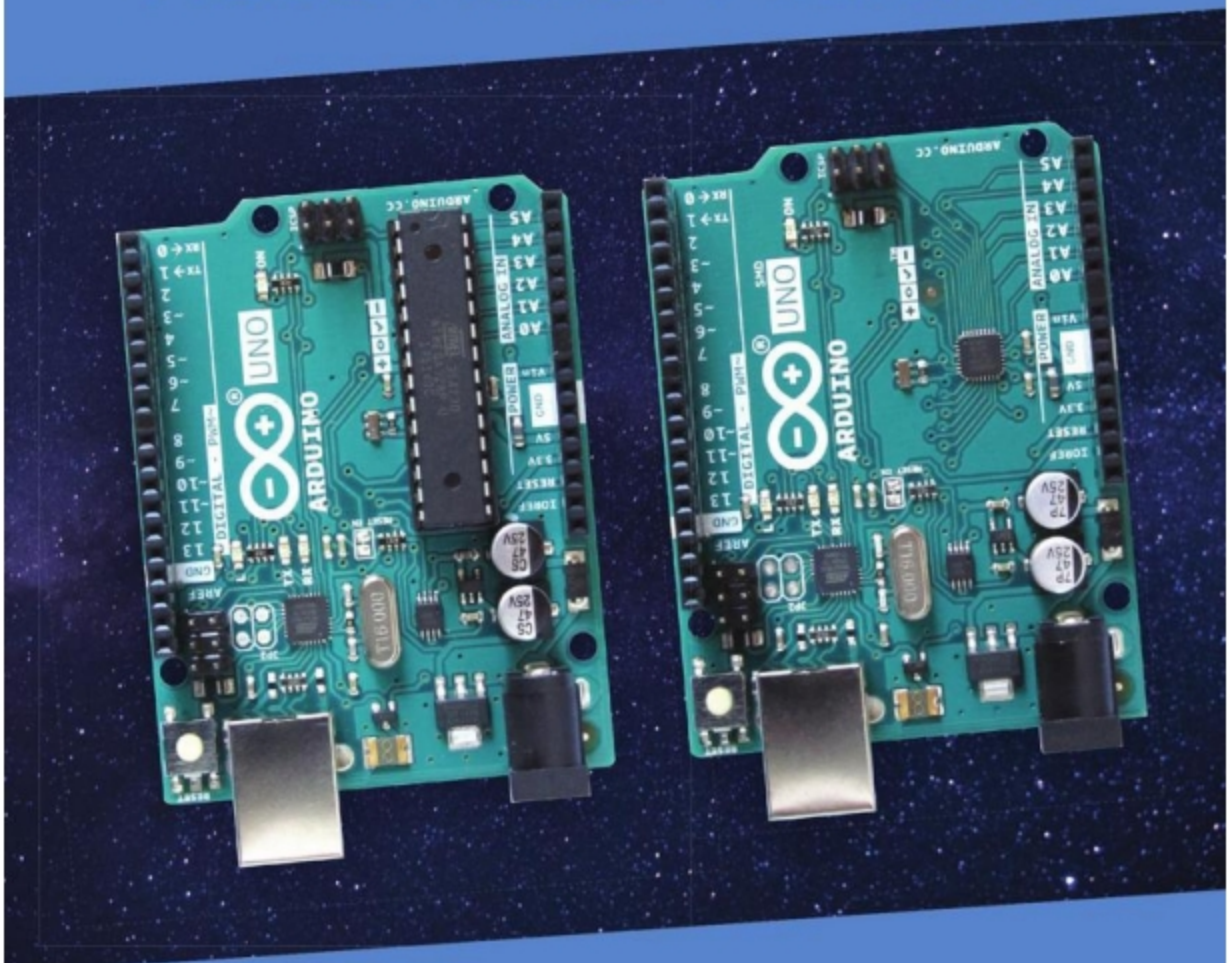
elektor books

Revised & Updated

A Reference and User Guide for the Arduino Uno Hardware and Firmware

ULTIMATE

Arduino Uno Hardware Manual



Warwick A. Smith

elektor
design › share › sell

This is an Elektor Publication.

Elektor is the media brand of Elektor International Media B.V.
PO Box 11, NL-6114-ZG Susteren, The Netherlands
Phone: +31 46 4389444

All rights reserved. No part of this book may be reproduced in any material form, including photocopying, or storing in any medium by electronic means and whether or not transiently or incidentally to some other use of this publication, without the written permission of the copyright holder except in accordance with the provisions of the Copyright Designs and Patents Act 1988 or under the terms of a licence issued by the Copyright Licencing Agency Ltd., 90 Tottenham Court Road, London, England W1P 9HE. Applications for the copyright holder's permission to reproduce any part of the publication should be addressed to the publishers.

Trademarks are property of their respective owners and are used in an editorial fashion with no intention of infringement of the trademark.

Declaration

The Author and the Publisher have used their best efforts in ensuring the correctness of the information contained in this book. They do not assume, and hereby disclaim, any liability to any party for any loss or damage caused by errors or omissions in this book, whether such errors or omissions result from negligence, accident or any other cause.

British Library Cataloguing in Publication Data

A catalogue record for this book is available from the British Library

ISBN: 978-3-89576-434-9 (print) ISBN: 978-3-89576-435-6 (e-book)

© Copyright 2021: Elektor International Media B.V.

Elektor is part of EIM, the world's leading source of essential technical information and electronics products for pro engineers, electronics designers, and the companies seeking to engage them. Each day, our international team develops and delivers high-quality content - via a variety of media channels (including magazines, video, digital media, and social media) in several languages - relating to electronics design and DIY electronics.
www.elektormagazine.com

Table of Contents

Introduction 13

Why Buy this Arduino Uno Hardware Manual? **13**

Target Audience **14**

Prerequisites **14**

Hardware Requirements **15**

Software Requirements **15**

What is Covered and What's Not Covered **16**

How to Use this Book **16**

Accompanying Resources **17**

Disclaimer, Errors and Corrections **17**

A Note from the Author **17**

Chapter 1 • Arduino Uno Overview 19

1.1 Arduino Uno Description and Functionality **20**

1.1.1 Arduino Uno 20

1.1.2 Arduino Uno SMD 20

1.1.3 Uses of the Arduino Uno 21

1.1.4 Arduino Uno Main Parts 22

1.1.4.1 Main Microcontroller 22

1.1.4.2 USB Connector 23

1.1.4.3 External Power In 23

1.1.4.4 Reset Button 23

1.1.4.5 Header Sockets with Arduino Pins 23

1.1.4.6 ON LED 24

1.1.4.7 L LED 24

1.1.4.8 TX LED (Transmit) 24

1.1.4.9 RX LED (Receive) 24

1.1.4.10 ATmega16U2 Microcontroller 24

1.1.4.11 ICSP Header 24

1.1.4.12 ICSP Header for ATmega16U2 25

1.1.4.13 Mounting Holes....26
1.1.5 Programming....26
1.1.6 Extending the Hardware....28
1.1.6.1 Add-on Boards: Shields....28
Stacking Shields....28
Shield Reverse Connection Protection....29
1.1.6.2 Prototype Shields....29
1.1.6.3 Strip-board....30
1.1.6.4 Electronic Breadboard....30
1.1.6.5 Custom PCB....31
1.1.7 Open-Source, Licensing and Logo....32
1.1.8 Third Party Compatible Boards....32
1.1.9 Build Quality, Warranty and Safety....33
1.1.10 Arduino and Genuino....34
1.2 Arduino Uno Firmware....35
1.2.1 USB Bridge Firmware....35
1.2.2 Bootloader....35
1.3 Precautions During Handling and Usage....36
1.3.1 Static Electricity....36
1.3.2 Work Surface....38
1.3.3 Power....38
1.3.4 Voltage....38
1.3.5 Handling....39
1.3.6 Interfacing Precautions....39
1.4 Arduino Uno History and Revisions....40
1.5 First Time Use and Basic Testing....41
1.5.1 New Arduino Uno Default Behavior....41
1.5.1.1 Computer Drivers....42
1.5.1.2 Arduino Uno Hardware Behavior....43
On LED....43
LED Flashed by Bootloader....43
Factory or User Loaded Sketch Running....43
1.5.2 Loading a Sketch to an Arduino Uno....43
1.5.2.1 Select Board and Port....44
1.5.2.2 Loading a Test Sketch....45
Open the Test Sketch....45

Upload the Sketch....45
Modify the Sketch....45
1.5.2.3 Serial Port Demonstration....46
Serial Port Sketch Example....46
Serial Monitor Window....46
1.5.3 Basic Testing....48
1.5.3.1 Visually Inspect the Board....48
1.5.3.2 Power LED and Voltages....48
1.5.3.3 Check Expected Default Behavior....49
1.5.3.4 Is the Board Recognized by the Host Computer?....49
1.5.3.5 Load a Test Sketch....49
1.6 Arduino Uno References and Help....49
1.6.1 Installing Software....50
1.6.1.1 Windows....50
1.6.1.2 MAC OS X....50
1.6.1.3 Linux....50
1.6.2 Getting Started, Examples and Reference....50
1.6.2.1 Getting Started Guides....50
1.6.2.2 Arduino Examples and Tutorials....50
1.6.2.3 Building Breadboard Circuits....50
1.6.2.4 Arduino Software Reference....50
1.6.3 Getting Help....51
1.6.4 Related Open-Source Projects....51
1.6.4.1 Fritzing....51
1.6.4.2 Wiring....52
1.6.4.3 Processing....52
1.6.5 Arduino Uno Boards on the Web....52
1.6.5.1 Arduino Uno Web Page....53
1.6.5.2 Arduino Uno SMD Web Page....53

Chapter 2 • Hardware Technical Information 55

2.1 Microcontroller....56
2.2 Atmel, Microchip and AVR....57
2.3 Memory....57
2.3.1 Flash Memory....57
2.3.1.1 Flash Memory Size....58

2.3.1.2 Flash Wear....58
2.3.1.3 Data Retention....58
2.3.2 SRAM....58
2.3.2.1 Volatile Memory....58
2.3.2.2 SRAM Size....58
2.3.3 EEPROM....59
2.3.3.1 EEPROM Programming....59
2.3.3.2 EEPROM Size....59
2.3.3.3 EEPROM Wear....59
2.3.4 Adding External Memory....59
2.3.4.1 SD Cards (SPI Interface)....59
2.3.4.2 Flash and EEPROM Chips (SPI / TWI Interface)....61
2.3.4.3 SPI Devices....61
2.3.4.4 TWI and I²C Devices....61
2.4 Power and USB....61
2.4.1 USB Power....62
2.4.2 USB Connection and Cable....62
2.4.3 External Power....63
2.4.4 Battery Power....64
2.4.5 Operating Voltage....66
2.5 Operating Frequency....66
2.6 LED Indicators and Reset Button....66
2.6.1 ON LED....67
2.6.2 L LED....67
2.6.3 TX LED....68
2.6.4 RX LED....68
2.6.5 Reset Button....68
2.7 User Pin Headers....68
2.7.1 Power Pins....69
2.7.1.1 GND Pins....69
2.7.1.2 5V Pin....69
2.7.1.3 3.3V Pin....69
2.7.1.4 Vin Pin....69
2.7.2 IOREF Pin and Unconnected Pin....69
2.7.3 RESET Pin....70
2.7.4 Digital and PWM Pins....70

2.7.4.1 Output Pins....70
2.7.4.2 Pin Current Rating....72
2.7.4.3 Input Pins....73
2.7.4.4 PWM Pins....74
PWM Example Sketch....75
Calculating PWM Duty Cycle....76
PWM Frequency....76
PWM LED Control Example....76
2.7.4.5 Serial Port / UART Pins....77
2.7.5 Analog In Pins....78
2.7.5.1 Analog In Example Sketch....78
2.7.5.2 Floating Analog Input Pin....78
2.7.5.3 Calculating Analog In Voltage....79
2.7.5.4 Analog In Pins Used for TWI....79
2.7.5.5 Analog In Pins Used as Digital I/O....79
2.7.6 AREF Pin....80
2.8 Programming Headers....81
2.8.1 Programming the Arduino Uno with External Programmer on ICSP....81
2.8.2 Restore the Bootloader with IDE and External Programmer....82
2.8.3 ATmega16U2 ICSP Header....82
2.8.4 ICSP Programming Resources....82
2.8.5 Using an Arduino as an In-System Programmer....83
2.9 Shared Pins....83
2.9.1 Serial Port Pins....83
2.9.2 L LED Pin....83
2.9.3 TWI or I²C Pins....83
2.9.4 ICSP SPI Pins and Reset....84

Chapter 3 • Pin Reference and Interfacing 87

3.1 Pin Default and Alternate Functions....88
3.1.1 Shared TWI Pins....88
3.1.2 Shared SPI Pins....88
3.2 ATmega328P to Arduino Uno Pin Mapping....90
3.2.1 ATmega328P Ports....92
3.2.2 ATmega328P Alternate Pin Functions....92
3.3 Pin Types and Interfacing....93

3.3.1 Digital Input / Output Pins....93
3.3.1.1 Pins as Outputs....93
Why an LED Needs a Series Resistor....94
How to Calculate a LED Current Limiting Series Resistor....95
Current Sourcing and Current Sinking....97
Current Sourcing....97
Current Sinking....98
Current Limitation Per Pin....99
I/O Port Current Source Limits....99
I/O Port Current Sink Limits....101
Switching Heavier Loads with Transistors and Relays....103
3.3.1.2 Pins as Inputs....105
Pull-down Resistor....106
Pull-up Resistor....107
Internal Pull-up Resistors....108
3.3.2 PWM Pins....109
3.3.3 Analog Pins....110
3.3.4 TWI Bus Pins....111
3.3.4.1 TWI Interfacing Example....111
Pull-up Resistors....112
3.3.4.2 Accessing TWI Devices in Software....113
3.3.5 SPI Bus Pins....114
3.3.5.1 SPI Bus Interfacing Example....114
3.3.5.2 Accessing SPI Devices in Software....116
3.3.5.3 Accessing SD Cards in Software....116
3.3.6 Serial / UART Pins....116
3.3.6.1 Hardware Serial Port....117
3.3.6.2 Software Serial Port....117
3.3.7 Power Pins....118
3.3.7.1 GND Pins....118
3.3.7.2 5V Pin....118
USB 5V....118
External Power to 5V Regulator....118
3.3.7.3 3.3V Pin....119
3.3.7.4 Vin Pin....119
3.3.8 Reset Pin....119
3.3.9 IOREF Pin....120

3.3.10 AREF Pin....120
3.4 ICSP Header on Main Microcontroller....121
3.5 ICSP Header on USB Microcontroller....122
3.6 JP2 Header on USB Microcontroller....123
3.7 Finding the Datasheets....125
3.7.1 ATmega328P Datasheet....125
3.7.2 ATmega16U2 Datasheet....125
3.7.3 Datasheets for Other Components....125

Chapter 4 • Power Reference 127

4.1 Power Supply Specification....128
4.1.1 Operating Voltage....128
4.1.2 USB Power Input....128
4.1.3 External Power Jack Input....128
4.1.4 External Power Jack Pinout....129
4.1.5 Vin Pin as Power Input....129
4.2 Power Circuit....129
4.2.1 External Power In and 5V Regulator....129
4.2.2 Power On Indicator LED....130
4.2.3 USB Power In....130
4.2.4 Automatic Switch....131
4.2.5 3.3V Regulator....131
4.2.6 Power Header Socket....131
4.3 Power Supply Protection....132
4.3.1 Reverse Polarity Protection....132
4.3.2 5V Regulator Protection Features....132
4.3.3 3.3V Regulator Protection Features....132
4.3.4 USB Overload Protection....132

Chapter 5 • Arduino Uno Firmware and Bootloader 133

5.1 Updating the USB to Serial Bridge Firmware using DFU....134
5.2 Microchip Studio....135
5.3 USB Microcontroller Firmware....137
5.3.1 Backing up the ATmega16U2 Firmware with Microchip Studio....137
5.3.2 DFU Bootloader Firmware....139

5.3.3 USB to Serial Bridge Firmware....139
5.3.4 Programming the USB Microcontroller using ICSP....140
5.3.5. ATmega16U2 Fuse Settings....141
5.4 Main Microcontroller Bootloader....142
5.4.1 Backing up the ATmega328P Firmware with Microchip Studio....142
5.4.2 Optiboot Bootloader Firmware....143
5.4.3 Restoring the Bootloader....144
5.4.4 ATmega328P Fuse Settings....144
5.5 The RESET-EN Solder Jumper....148
5.6 Alternative Firmware Programming Methods....149

Chapter 6 • Circuit Diagram and Components 151

6.1 Circuit Diagram....152
6.1.1 Block Diagram....152
6.1.2 Main Microcontroller Circuit....153
6.1.3 USB Microcontroller Circuit....154
6.1.4 Power Supply Circuit....155
6.2 Component List....156
6.3 Component Positions on the Board....159
6.4 Getting an Electronic Copy of the Circuit Diagram....160

Chapter 7 • Fault Finding and Measurement 161

7.1 Basic Fault Finding and Repair....162
7.2 Replacing the Main Arduino Uno Microcontroller....162
7.2.1 Replacement Part....163
7.2.2 Removing the Old Microcontroller....163
7.2.3 Inserting the New Microcontroller....163
7.2.4 Loading the Bootloader....165
7.3 Voltage Measurements....165
7.3.1 Powered from USB....166
7.3.1.1 5V Test Points....166
7.3.1.2 USB Power and 3.3V Circuit Test Points....166
7.3.1.3 Measuring Voltage on the L and ON LEDs....167
7.3.2 Powered from External Power Supply....167

7.4 Waveform Patterns and Measurement....169
7.4.1 Testing for Presence of Microcontroller Clocks....170
7.4.2 Testing for PWM Waveforms....171
7.4.3 Testing the UART Output....172
7.4.4 TWI Signals....173
7.4.5 SPI Signals....175

Chapter 8 • Mechanical Dimensions and Templates 177

8.1 Measurements, Tolerance and Scale....178
8.2 Length, Width and Mass....178
8.3 Mounting Hole Spacing and Size....180
8.4 Shape Dimensions....181
8.5 Header Positions....182
8.6 Shield Reverse Connection Protection....183
8.7 Using Strip-board as a Shield....184
8.8 Drill Template and KiCad Template....184

Appendix A • Specifications Quick Reference 185

Index 189

Introduction

Welcome to the Arduino Uno Hardware Manual, a reference and user guide for the Arduino Uno hardware and firmware. This manual provides up to date hardware information for the popular Arduino Uno, the easy to use open-source electronics platform used by hobbyists, makers, experimenters, educators and professionals.

Why Buy this Arduino Uno Hardware Manual?

There are four main reasons to buy this book besides just wanting a handy reference manual for the Arduino Uno hardware and firmware for use on the workbench. These reasons are discussed below.

1) All the Hardware Information in One Place

Although there is a lot of information about the Arduino Uno hardware available online, it is spread out over many pages and websites. Having all of the Arduino Uno hardware information in one place, in this concise guide, is convenient and a time saver. This is especially true for new Arduino users who may not even know what to look for when getting familiar with the Arduino Uno hardware, for example, how to extend the hardware, how to add external data memory, how to connect hardware to Arduino pins in current sinking and current sourcing configuration. It also includes practical information such as how to replace the main microcontroller, which firmware needs to be loaded to a new microcontroller and which other settings must be changed for it to work. In addition, this manual has detailed hardware technical information, a pin reference chapter, basic interfacing information, a power reference chapter, information on all firmware programs loaded on an Arduino Uno, the circuit diagram and component list, fault finding and testing steps, as well as hardware mechanical dimensions and measurements.

2) Integrity of Information

Information on the internet about Arduino or any other subject may or may not be technically correct. Discerning which information is correct and which is incorrect is a problem, especially for those users who are new to electronics. Even hardware specifications from Arduino themselves are often badly specified or even incorrectly specified. This manual provides hardware information on the Arduino Uno that has been carefully checked and verified to be correct.

3) Presentation of Information

Each topic or subject in this book has been broken down into smaller parts and carefully explained and described. For this reason, each chapter contains many sections, each with its own sub-heading and section number for easy reference and cross-reference. Numerous illustrations and figures are included for easier understanding of explanations and for quick and easy reference.

4) Information Not Available Elsewhere

This book contains some information that is not available online, or is hard to find. It also offers unique views and illustrations of the hardware that reveal some interesting aspects of the Arduino Uno hardware that is not obvious when looking at the circuit diagram or other sources of information.

Target Audience

This manual has been written for anyone interested in the Arduino Uno who would like an easy to use hardware reference, including hobbyists, makers, experimenters, teachers, students, and professionals such as electronic engineers.

Prerequisites

It is assumed that the reader of this manual has at least used the Arduino Uno on a basic level and has some basic knowledge of electronics. This is not a book that explains the basics of electronics, but assumes that the reader already has the knowledge to understand terms such as voltage and current, and knows what electronic components such as resistors and transistors are. In any case, this is the basic knowledge that is required to use an Arduino. Those readers who do not have this basic knowledge must at

least be willing to learn, by using additional resources, be it via the internet or by using other books on the subject.

Hardware Requirements

At a minimum, an Arduino Uno will obviously be required, as well as a computer with which to program the Arduino board. This book mainly references the Arduino Uno revision 3 (REV3 or R3) board which is the latest Arduino Uno at the time of writing. It also mentions earlier versions of the Arduino Uno when necessary and makes some reference to clone or compatible boards. For advanced use and programming firmware to the Arduino Uno, one of several USB programmers can be used – see the text for more details, but most users will not need to do this, unless updating firmware on a new microcontroller, and even then there are cheaper alternatives such as using one Arduino to update the firmware on a second Arduino.

In the pin reference and interfacing chapter, an electronic breadboard and jumper wires are used to show how to interface LEDs, transistors, a relay, an external TWI memory chip and an SD card adapter. These are just interfacing examples and users will only need the components for a particular project or interfacing example that they are interested in. Hardware testing requires at least a multimeter. Some tests require an oscilloscope, but it is not essential for every user and is only used in a small section of the book.

Software Requirements

Without software, an Arduino Uno does absolutely nothing. For this reason, although this is a hardware manual, the free Arduino IDE programming environment must be installed. This enables Arduino programs or sketches to be loaded to the Arduino Uno to operate and test the hardware.

All software used in this book is either free and open-source, such as the Arduino IDE software, or free software that is not open-source. All software used can easily be downloaded from the internet at no cost. Other software, besides the Arduino IDE, is optional and will only need to be installed if required for certain tasks.

What is Covered and What's Not Covered

This book is primarily about hardware on an Arduino Uno board, including technical information, a pin reference, power supply information, firmware details, circuits and parts list, fault finding and measurement, as well as mechanical information, but has to include some software, in the form of user programs or sketches, in order to get the hardware to work. That being said, it does not cover software programming in great detail, but does provide references to the appropriate example sketches in the Arduino IDE and online documentation where necessary. Most of the functionality of an Arduino Uno comes from the main microcontroller on the board. This microcontroller is covered by the text in the context of Arduino and how it is set up and used by the Arduino IDE and its libraries. Advanced use of some of the internal hardware in the main microcontroller that is not available in the Arduino IDE and libraries is not covered, except to provide a pin reference for these advanced features that shows which Arduino pins are mapped to advanced hardware. Users who want to use the advanced hardware that is not available in the Arduino IDE or libraries must reference the datasheet for the main microcontroller.

How to Use this Book

It is suggested that this book is first read through once to get a good idea of its contents and a better understanding of the Arduino Uno hardware, it can then be used as a reference manual. Although this book has been written as a manual, it is not dry like some manuals or textbooks. Anyone interested in Arduino should find it an enjoyable read.

For electronic engineers or competent hobbyists who have used other microcontroller boards, but not an Arduino Uno, this book provides an easy introduction to the Arduino Uno hardware that will get new users started quickly with the board. The first two chapters of the book are basically a hardware user manual, while the remaining chapters are more like a reference manual, although there is some overlap between the two.

When using the book for reference, use the index to find specific topics and use appendix A to quickly find technical specifications. Appendix A has a cross reference for each technical specification that references the same specification in the main text, which provides detailed information. Each section in the book cross references related material in other sections and chapters. The table of contents is also useful for referencing information, as it is a list of every section in the book.

Accompanying Resources

Accompanying resources can be found on the Elektor (www.elektor.com) web page for this book, and on the author's supporting website (wspublishing.net) which includes links to all of the online resources so that they do not need to be typed in manually. Arduino software sketches used in the book are also available from the supporting website.

Disclaimer, Errors and Corrections

Although this manual is not an official Arduino manual, or endorsed by Arduino in any way, it has been written and thoroughly checked by a competent electronic engineer. The information in this manual has been carefully checked and proof read, but this is not a claim of infallibility, and this manual, like any work, is subject to typos and small errors that may have been missed during the proof reading process. In the event that you do find an error or typo, please use the accompanying website (wspublishing.net) to report it. This will be a big help to others as any corrections will be published on the supporting website.

A Note from the Author

Thank you for purchasing this manual! I hope that you thoroughly enjoy reading this book and using it as a reference. Please give an honest review of this book on the web page for this book at the Elektor website (www.elektor.com). This will provide feedback to the author that can be used to improve future versions of this manual. When writing a review, bear in mind that this book is an Arduino Uno hardware manual and not a software or electronics book. Review the book contents in light of its title and stated purpose. Also bear in mind that some information in this book is repeated because of the layout of the book – with chapter 1 being an overview of the Arduino Uno, chapter 2 containing technical information and chapter 3 a pin reference, some information is repeated, but more details are added in each chapter. The alternative to this is to lump everything related together, in which case the book can not be laid out in an easier to read format that builds the information up from a top level view of the hardware followed by more details on each aspect of the hardware.

Chapter 1 • Arduino Uno Overview

Before taking a closer look at each aspect of the Arduino Uno hardware in greater depth, it is necessary to get a top level overview and general understanding of the Arduino Uno. This chapter, together with chapter 2, can be thought of as a "hardware user manual" for the Arduino Uno and covers all of the basics that are needed before a more comprehensive study of the Arduino Uno hardware that is contained in the chapters that follow.

Included in this chapter is a general overview of the Arduino Uno, its main parts, and how to extend its hardware.

Basics of the firmware, the software that comes factory loaded on a new Arduino Uno board, is explained. The chapter wraps up with board handling precautions, a brief history of the Arduino Uno and the hardware revisions that it has been through, some information on first time use, and basic testing.

In this Chapter

- A top level view of the Arduino Uno hardware and functionality
- A brief look at the Arduino Uno firmware and what it does
- Board handling and usage precautions
- Arduino Uno history and revisions
- First time use and basic testing
- References to quickly find information on the Arduino Uno

1.1 Arduino Uno Description and Functionality

The Arduino Uno is a single board computer that uses a microcontroller as its main processor to run software loaded via the Arduino IDE or other programming environment. It comes in two different versions, namely, the Arduino Uno and Arduino Uno SMD. These boards are identical except for the physical package of the main microcontroller.

1.1.1 Arduino Uno

Most users will want to get the Arduino Uno (Figure 1.1) rather than the Arduino Uno SMD board because it has a socket that houses the main microcontroller. This allows easy replacement of the microcontroller should it be damaged or bricked up by software. The main microcontroller is the 28-pin chip that can be seen above the text “POWER” and “ANALOG IN” in Figure 1.1.

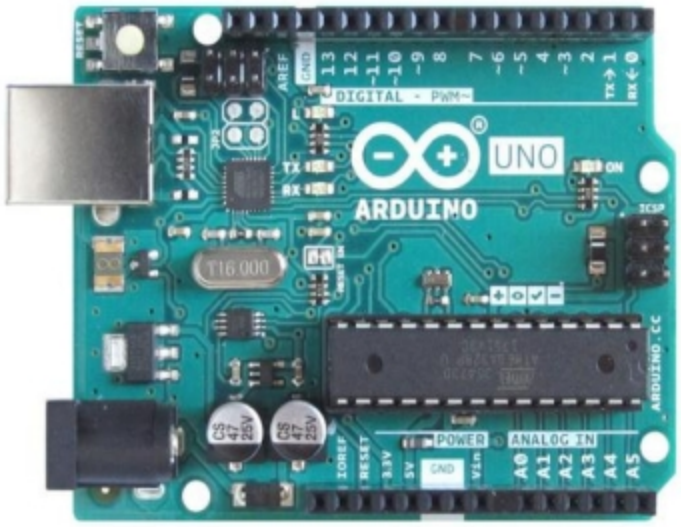


Figure 1.1: Top View of an Arduino Uno

1.1.2 Arduino Uno SMD

On the Arduino Uno SMD board, the main microcontroller is a surface mount device (SMD) which can be seen above the word “POWER” in Figure 1.2. This means that it can not easily be replaced if it is ever damaged, as it is directly soldered to the top of the board. It is almost impossible to replace this microcontroller without using a professional soldering rework station.

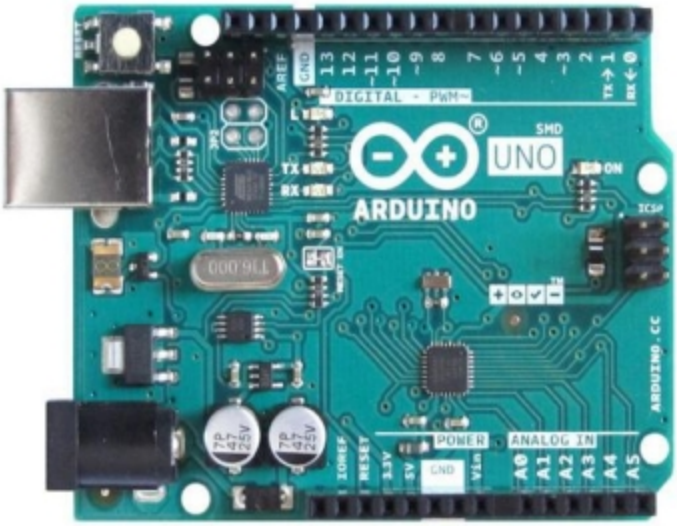


Figure 1.2: Arduino Uno SMD Board

According to the Arduino website, the Arduino Uno SMD board was made in response to a shortage in supply of the through-hole version of the microcontroller. Through-hole here refers to the way the microcontroller is mounted on the Arduino Uno board of Figure 1.1 where the pins of the microcontroller socket go through holes in the Arduino board and are soldered underneath. The through-hole microcontroller then plugs into the socket.

1.1.3 Uses of the Arduino Uno

Arduino is based on the open-source Wiring project (wiring.org.co) which was originally intended for art and design students who wanted to use electronics in their projects. Originally the Wiring board and the first Arduino (before the Arduino Uno) were used in education for student projects. Arduino has become popular beyond its original design purpose and is now used by everyone from hobbyists, makers, hackers and educators to engineers and other professionals.

Some examples of Arduino projects are robots (many different kinds, including drawing robots and plotters), colored light sequencers and other light displays, clocks, electronic games, devices for measuring temperature and humidity, vending machines and many more. A good place to look for Arduino projects, other than doing an internet search, is at the Arduino Project Hub (create.arduino.cc/projecthub) where projects that use the Arduino

Uno can be filtered using the products selection menu. Projects can also be filtered by category, difficulty and type.

Other uses of an Arduino Uno are for quick prototyping and testing of electronic components. For example, a temperature sensor can easily be wired to an Arduino Uno and tested by writing a simple software sketch using the Arduino software programming environment (called the Arduino IDE). Temperature readings can then be displayed in the Serial Monitor window of the Arduino IDE to see whether the sensor is working.

1.1.4 Arduino Uno Main Parts

Figure 1.3 shows the main parts of an Arduino Uno that will be of interest to a normal user of the board. Each of these parts is described in more detail in the sections that follow.

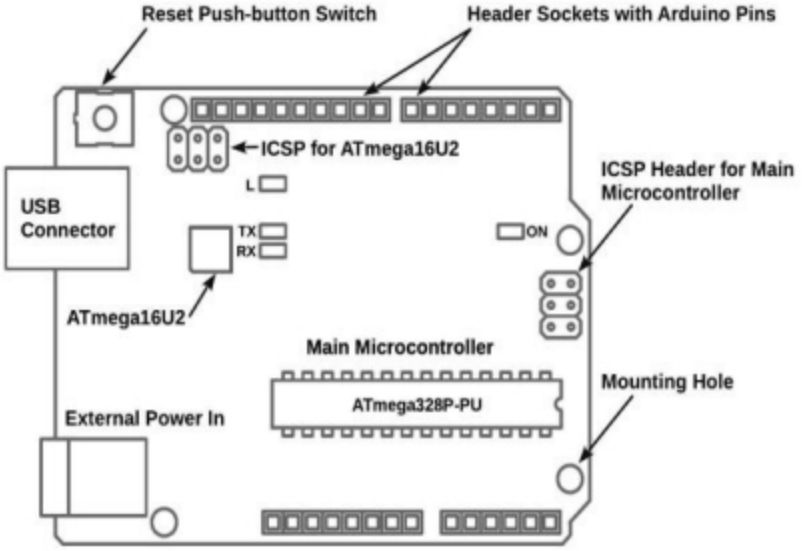


Figure 1.3: Main Parts of the Arduino Uno

1.1.4.1 Main Microcontroller

The main microcontroller (part number ATmega328P) runs sketches, or user programs, that are loaded to the Arduino from the Arduino IDE programming environment. Section 2.1 of the next chapter has more information on the main microcontroller of both the Arduino Uno and Arduino Uno SMD boards.

1.1.4.2 USB Connector

The USB connector can be used to both power the Arduino Uno and program it. USB programming is the default method of programming an Arduino Uno when using the Arduino IDE software. The USB connector is a standard type B connector. Use an A to B USB cable for connecting the Arduino Uno to a computer.

A user program, called a sketch, can use the USB connection to send and receive serial data. This allows user programs to send data to, and receive data from a computer. The Arduino IDE has a utility called the Serial Monitor, which is one method that can be used to send and receive data over the USB link. Alternatively other serial terminal programs or custom software running on a computer can use the same link to communicate with the Arduino via USB.

1.1.4.3 External Power In

External DC (Direct Current) power with a voltage of between 7V to 12V can be applied to the external power connector of the Arduino Uno, allowing it to run stand-alone without the need of a computer connected to the USB port. When external power is supplied to the Arduino Uno and the USB cable is plugged in for programming, the Arduino will automatically switch to using external power.

External power can be in the form of a DC power supply that converts mains power to low voltage DC, for example a “wall wart” power supply. Alternatively, a battery can be used, as long as the voltage is in the correct range of 7 to 12V. When connecting external power it is very important to get the polarity of the supply connected to the Arduino correctly. The center pin of the external power connector is positive on the Arduino Uno.

1.1.4.4 Reset Button

A reset button next to the USB connector allows manual resetting of the Arduino. Pressing the reset button will reset the main microcontroller, causing the software that is loaded on the main microcontroller to start running from the beginning.

1.1.4.5 Header Sockets with Arduino Pins

Header sockets are found on either side of the Arduino Uno, along its length. Each hole in the socket is usually referred to as an Arduino pin. There are several types of pins, including power pins, analog input pins, digital pins and a few special pins. Section 2.7 of chapter 2 has more information on the pin types and functions.

1.1.4.6 ON LED

When power is supplied to the Arduino Uno, either from the USB connection, or from an external power supply, the ON LED will turn on, indicating that the Arduino Uno is powered up.

1.1.4.7 L LED

The L LED is a user programmable LED connected to digital pin 13 on an Arduino Uno. In other words it can be switched on and off by instructions or statements in any user sketch by controlling digital pin 13.

1.1.4.8 TX LED (Transmit)

When data is transmitted from the main microcontroller over the serial to USB link, the TX LED will flash or blink.

1.1.4.9 RX LED (Receive)

When data is received by the main microcontroller over the serial to USB link, the RX LED will flash or blink.

1.1.4.10 ATmega16U2 Microcontroller

An ATmega16U2 microcontroller in a tiny surface mount package can be found near the USB connector (see Figure 1.3). This microcontroller acts as a bridge between the USB connection and the main microcontroller for either programming the main microcontroller, or when a user sketch sends and receives serial data over the serial to USB link.

This microcontroller comes with factory loaded software, usually referred to as firmware, that gives it its functionality as a USB bridge. For most normal use, this microcontroller is never programmed by the end user.

1.1.4.11 ICSP Header

Most Arduino users won't need to use the ICSP header, except in the case where an extension board is used with the Arduino that happens to connect to the ICSP header. Some advanced operations are available through the ICSP header and are described further in this section. New Arduino users may just want to browse through the rest of this section to see what the ICSP capabilities are, but it is not necessary to know about them for normal use. ICSP stands for In-Circuit Serial Programming.

The ICSP header is found at the end of the board, on the opposite side to the USB connector, and is wired to the main microcontroller. This header has two main uses. It is used by some extension boards (known as shields), such as Ethernet shields which have a connector under them that connects to this header in addition to the normal header sockets down the sides of the board. The second use of the ICSP header is to connect an external USB programmer such as an Atmel-ICE, or any number of homemade or third party programming devices. An external programmer can be used to program the Arduino Uno through the ICSP header as an alternative to using the default USB programming. This allows the full capacity of the program memory to be used – factory loaded bootloader software that is necessary for USB programming to work uses up some of the program memory (where sketches are loaded to). When using an external programmer, software sketches overwrite the bootloader and make use of the memory that it was occupying.

External programmers also allow the internal programmable fuse settings of the main microcontroller to be easily changed using software such as Microchip Studio (which used to be called Atmel Studio, but was rebranded to Microchip Studio). For normal use these fuses are left at their Arduino factory set values. A new bootloader can easily be programmed to the main microcontroller using an external programmer. This means that a board on which the bootloader was overwritten, after using an external programmer to load user sketches, can have its bootloader restored by an external programmer. The board will then be able to use USB programming again without the need of an external programmer.

Alternative programming tools such as Microchip Studio and others can be used to program the Arduino using plain C or C++. Some external programming tools that connect to the ICSP header have debugging capabilities, and when used in combination with Microchip Studio or similar tools, allow single-step debugging and examination of the internal registers and variables of the main microcontroller. Those interested in learning the C language to program Arduino boards may be interested in the book *C Programming with Arduino ISBN 978-1-907920-46-2* by the same author, also published by Elektor.

1.1.4.12 ICSP Header for ATmega16U2

A second ICSP header can be found near the USB connector that connects to the ATmega16U2 microcontroller. Most Arduino users will never need to use this header.

This header can be used to load software to the ATmega16U2 USB bridge microcontroller using the same tools described for the main microcontroller ICSP header. It can be used to

update the ATmega16U2 firmware or for advanced users who are developing their own firmware for this microcontroller. An alternative to using this ICSP header to update the firmware on the ATmega16U2 is available, called DFU (Device Firmware Update) programming. DFU programming allows the ATmega16U2 to be programmed using the USB connection.

1.1.4.13 Mounting Holes

Four mounting holes can be found near the edges of the Arduino Uno. These holes can be used to mount the Arduino to a base plate or enclosure using screws, or bolts and PCB stand-offs.

1.1.5 Programming

Arduino programming software is available for Windows, Mac OS X and Linux, and is known as the Arduino IDE (Integrated Development Environment). Although this is a manual on the Arduino Uno hardware, it is still necessary to have the Arduino IDE programming software installed and ready, as it is used to run various programs for testing the hardware and for demonstrating how the hardware works. Without software, the hardware of the Arduino Uno does nothing.

User written programs that are typed into the Arduino IDE are known as *sketches*. Sketches are the equivalent of source code files found in other programming languages. Sketches are written in the editor window of the Arduino IDE where they can be verified to see if they contain valid code by clicking the Verify button. Valid sketches can be uploaded to the Arduino using the Upload button in the Arduino IDE. Whenever the Verify or Upload button is clicked in the Arduino IDE, the build process of the sketch is started, which goes through preprocess, compile and link steps to convert the sketch into a file usable by the main microcontroller.

The Arduino IDE is free to download from www.arduino.cc/en/software where the correct download must be selected for the desired operating system. For Windows, there are three options for installing the Arduino IDE: by downloading and running the Windows installer, from the Microsoft Store, or via a ZIP file. If using a Linux or Apple computer, refer to section 1.6.1 near the end of this chapter for links to online installation guides.

It is suggested to just download the ZIP file and extract the contents of the ZIP file (the folder found inside) to a convenient location, such as the desktop. Figure 1.4 shows the

contents of the folder that was extracted from the ZIP file and then copied to the Windows desktop. Make sure to extract the folder from the ZIP file and not just open the ZIP file – first open the ZIP file and then drag and drop the folder from the ZIP file to the desktop.

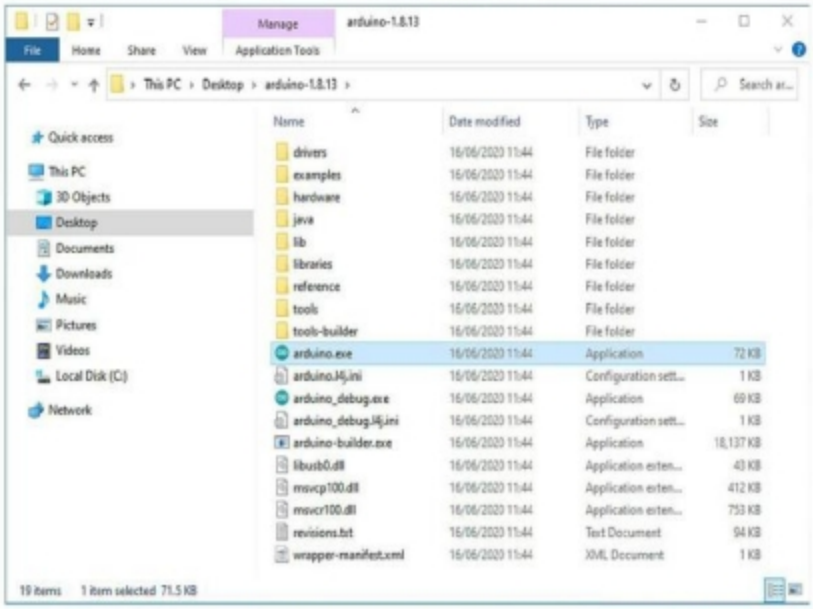


Figure 1.4: Arduino IDE Software Unzipped to a Folder on the Desktop

For those readers who may have already installed the Arduino IDE software using one of the other methods, this is not a problem. Having the Arduino software folder on the desktop does offer some advantages, but is mostly the preference of the author. One advantage is that it allows multiple versions of the IDE to be available on a single computer. It is possible that a newer version of an IDE can cause sketches of an older project to stop working. Building the same project using an older version of the IDE can resolve the problem. Although this type of software bug occurs rarely, it can and does still happen, hence the advantage of having different versions of the Arduino IDE on the same computer. For the purposes of this book, having the Arduino IDE software in an easy to find location on the desktop is convenient when we look at the firmware files that are located in this folder, rather than trying to find them in the file system. Users who have already installed the Arduino IDE using one of the other methods still have the option of downloading the ZIP file for the purpose of examining the software and easily finding the firmware files.

To start the Arduino IDE from the folder extracted from the ZIP file, simply open the folder and double-click the Arduino application file to run it. The Arduino application file can be seen highlighted in Figure 1.4. To remove the software from the computer, just delete the Arduino IDE folder. When a newer version of the software becomes available, the old version can either be left in place or deleted.

Alternative programming environments to the Arduino IDE are available, such as Microchip Studio, other free and commercial professional programming tools, and command line tools. This book uses the Arduino IDE software as the main method of programming the Arduino Uno. The Arduino IDE is the main or default way of programming Arduino boards, especially for new users.

1.1.6 Extending the Hardware

There are several ways to extend the Arduino hardware including: buying add-on boards, using prototype boards, building circuits on strip-board, using an electronic breadboard and making a custom PCB (Printed Circuit Board). A description of each of these methods for extending Arduino Uno hardware can be found in the sections that follow. Ultimately whichever method is used to extend the hardware, they all make use of the connector sockets found on each side of the Arduino Uno down its length. Hardware can be connected directly to any of the Arduino pins from the connector sockets, or use specific pins such as those that provide serial bus interfaces like SPI or TWI. SPI and TWI are explained further in section 2.3.4 – Adding External Memory, in the next chapter.

1.1.6.1 Add-on Boards: Shields

External add-on boards called *shields* that plug into the header sockets of the Arduino Uno are used to extend the hardware of the board. Shields are available from Arduino as well as third party vendors, and include such functionality as Ethernet, WiFi, motor control, USB host support, relays, data logging and more.

Stacking Shields

Shields can be stacked one on top of the other to add multiple hardware functionality. An example of an Ethernet shield plugged into an Arduino Uno board is shown in Figure 1.5 at the left. A stack of two shields is shown at the right of the figure where the first shield is plugged into the Arduino Uno and a second shield is plugged into the first shield.

When connecting more than one shield to an Arduino, it is important to know which pins of the Arduino connectors are used by the shields. In some cases the pins can be shared, such as when the pins are used as a serial bus such as SPI or TWI. In other cases the shields may not be compatible if they both use one or more of the same pins.

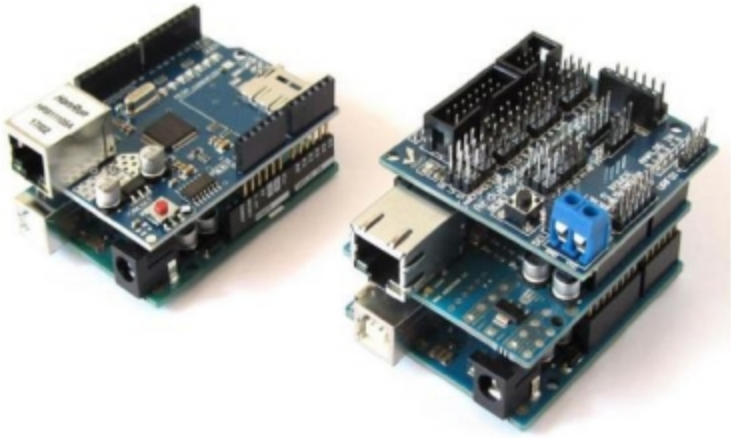


Figure 1.5: Arduino Uno with Ethernet Shield Plugged in (left) and Shields Stacked on an Arduino Uno (right)

Shield Reverse Connection Protection

To prevent shields from being inserted into an Arduino the wrong way around, one of the Arduino headers is spaced so that it is slightly offset from the headers on the other side of the board. This is essential for a classroom environment where hardware could easily be destroyed by students plugging in shields the wrong way around. For more information on the spacing of the headers on an Arduino Uno, see Chapter 8, section 8.6.

1.1.6.2 Prototype Shields

Besides ready made and assembled shields, prototype shields are also available that allow custom circuits to be built on them by soldering the desired components to the board. These prototype shields are available from Arduino and from third party manufacturers.

Figure 1.6 shows two examples of blank prototype shields. At the left of the figure is a third party prototype shield, and at the right is a Proto Shield Rev3 from Arduino, part number TSX00083 (store.arduino.cc/proto-shield-rev3-uno-size), which includes long pin

connectors. As can be seen in the figure, prototype shields are blank circuit boards, to which connectors can be soldered so that they will plug into an Arduino Uno like a shield does. They have holes and tracks to which components can be soldered to build a circuit.

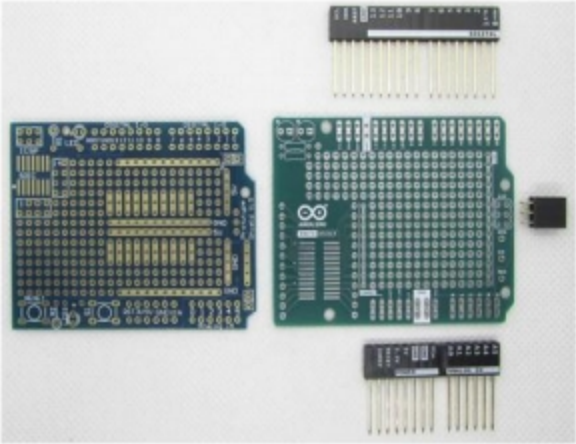


Figure 1.6: Prototype Shields for Arduino Uno

1.1.6.3 Strip-board

Strip-board, such as Veroboard, that has been available to electronic hobbyists for many years can be used to make custom shields. The main problem with strip-board is that one of the Arduino Uno headers is offset from the headers on the opposite side of the board, which means that the holes in the strip-board won't line up with the holes in the offset Arduino header, see chapter 8, sections 8.6 and 8.7 for more details. One option that allows strip-board to be used with an Arduino Uno is to not use any pins from the offset header. Another option is more of a hack where long pins are soldered in the strip-board and bent to fit into the offset header.

1.1.6.4 Electronic Breadboard

Temporary circuits can be built on electronic breadboards which allows electronic components to be plugged into and removed from them. Breadboard circuits can be connected to the Arduino Uno pins via the header sockets using jumper wires, as can be seen in Figure 1.7, which shows two LEDs with series resistors connected to digital pins of an Arduino Uno. A software sketch can be written to control the two LEDs in the circuit, switching them on and off. Components and wires can be unplugged from a breadboard and reused, which is ideal for prototyping and educational use.

In addition to discrete electronic components, various modules are available that can also be plugged into a breadboard to build circuits. Packs of modules are available that include, for example, 30 modules of various types, such as LED modules, light sensors, tilt switches, temperature and humidity sensors, relays and others. Each module consists of a small circuit board with components soldered to it to make up the circuit. Modules have pins that are used to plug them into a breadboard.

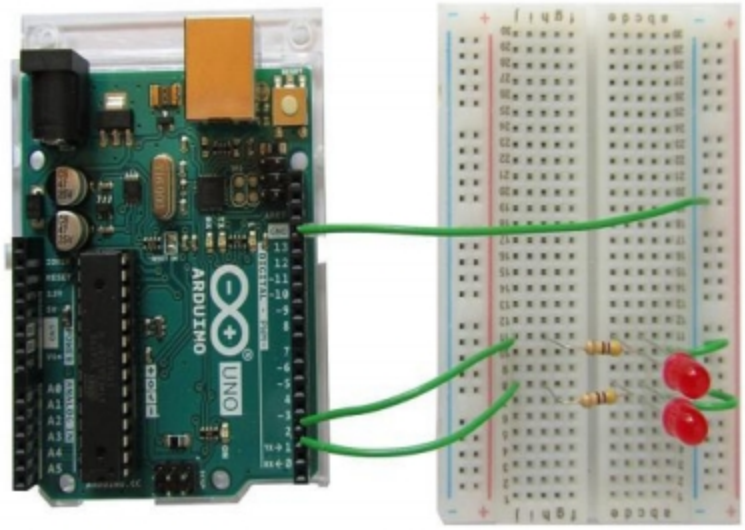


Figure 1.7: A Simple Breadboard Circuit Interfaced to an Arduino Uno

1.1.6.5 Custom PCB

A shield in the form of a custom made PCB (Printed Circuit Board) can be designed and built. This method is more suited to those who have some experience with electronics, as it requires some circuit design skills and ability to use a suitable electronic CAD (Computer Aided Design) package. Also see Fritzing in section 1.6.4.1 in this chapter.

Options here are to either etch a board at home or in a work laboratory, or send the PCB design out to be professionally manufactured. KiCad (kicad.org) is an open-source EDA (Electronic Design Automation) software package that can be used to draw circuit diagrams and lay out PCBs. A KiCad template for an Arduino shield is provided with this book – see Chapter 8, section 8.8 for details. Figure 1.8 shows an example of a custom homemade Arduino shield that was designed using KiCad.

Figure 1.8: Homemade Custom PCB Shield

1.1.7 Open-Source, Licensing and Logo

Arduino boards are open-source hardware which means that the design files for the hardware are available to examine and modify by anyone. The design files are licensed under a Creative Commons Attribution Share-Alike license which allows personal and commercial derivative works, but they must credit Arduino and release the derivative works under the same license. See creativecommons.org/licenses for more information on the various Creative Commons licenses.

The Arduino IDE software is also open-source which means that anyone can have access to the source code which can be examined and modified. Source code for the Java environment (used by the Arduino IDE software) is released under the GPL (General Public License). The C/C++ libraries are released under the LGPL (Lesser General Public License).

Arduino logos are not open-source, but are trademarks of Arduino. Use of the Arduino name is restricted and Arduino logos may not be copied. For more details, see the page www.arduino.cc/en/trademark on the Arduino website.

1.1.8 Third Party Compatible Boards

Because Arduino is open-source, many third party Arduino boards are available including third party clone boards and boards that are compatible with the Arduino Uno. Figure 1.9

shows a board that is compatible with the Arduino Uno. Although this board works like an Arduino Uno, it has some slight differences such as the physical package that the main microcontroller is housed in, which differs from both the Arduino Uno and Arduino Uno SMD boards manufactured by Arduino. Some other differences are the layout of components and the color of the various indicator LEDs on the board. Additionally this board does not use an ATmega16U2 for the USB bridge chip, but uses a cheaper alternative chip. Clone or compatible boards may offer some additional features to the official Arduino boards, such as the extra holes in the board of Figure 1.9 that can be used to solder pins, connectors or wire for the purpose of connecting circuits to the Arduino pins.

Third party Arduino clones and Arduino compatible boards vary in quality from excellent to poor quality. Some of the cheaper boards, usually from China, are quite usable, but there is always the risk of getting a dud. Note the Chinese board in Figure 1.9 has the ICSP header at the right incorrectly labeled “ISCP”. For more information on the various spin-off boards including official boards, clones, derivatives, compatibles and even counterfeits, see the article blog.arduino.cc/2013/07/10/send-in-the-clones/ on the Arduino blog.

Figure 1.9: Arduino Uno Compatible Board from a Third Party Manufacturer

1.1.9 Build Quality, Warranty and Safety

Buying an official Arduino board manufactured by Arduino or one of its authorized manufacturers guarantees that the board will be a quality product, and that some of the money paid for the board will go back into supporting the Arduino community, such as for

further hardware development, documentation and hosting of the support forums. A full list of official Arduino boards can be found at www.arduino.cc/en/Main/Products on the Arduino website.

For more information on Arduino board FCC compliance and for the warranty statement, see www.arduino.cc/en/Main/warranty where probably the most important clause is with regards to safety-critical applications. This clause (clause 1.5) is included below, verbatim as it was found on the Arduino website at the above link at the time of writing.

" 1.5 Arduino LLC products are not authorized for use in safety-critical applications where a failure of the Arduino LLC product would reasonably be expected to cause severe personal injury or death. Safety-critical applications include, without limitation, life support devices and systems, equipment or systems for the operation of nuclear facilities and weapons systems. Arduino LLC products are neither designed nor intended for use in military or aerospace applications or environments, nor for automotive applications or the automotive environment. The Customer acknowledges and agrees that any such use of Arduino LLC products is solely at the Customer's risk, and that the Customer is solely responsible for compliance with all legal and regulatory requirements in connection with such use. "

1.1.10 Arduino and Genuino

When starting the Arduino IDE software, the splash screen shows both the name Arduino and *Genuino*. The reason for the Genuino name is because of a trademark dispute. The dispute has since been resolved.

A dispute over the trademark "Arduino" caused Arduino to split into two companies. A new Arduino company was started with a new website at **arduino.org** and the original Arduino company with the original **arduino.cc** website. The folks at arduino.cc owned the Arduino trademark in the USA, but not outside of the USA. They then came up with the name *Genuino* to use for boards that sold outside the USA, in order to sidestep the trademark issue, and the name *Arduino* for boards sold in the USA. Genuino was just a re-branding in order for the original team to be able to sell Arduino boards outside the USA. The whole legal battle has since been resolved and there is now only one Arduino website – arduino.cc and they own all of the trademark rights to the Arduino brand name.

1.2 Arduino Uno Firmware

As explained earlier in this chapter, the Arduino Uno has two microcontrollers – one that acts as a USB to serial bridge and the other is the main microcontroller that runs user loaded programs or sketches. Each of these microcontrollers comes with factory loaded software, usually referred to as firmware in embedded systems such as Arduino.

1.2.1 USB Bridge Firmware

Firmware in the ATmega16U2 chip sets this microcontroller up as a virtual serial port, which is usually referred to as a virtual COM port on a Windows computer. It acts as a bridge between the USB port of the Arduino Uno and the serial port or UART of the main microcontroller. When the serial port driver loads on the host computer after plugging the Arduino Uno into a USB port, the Arduino looks like a serial port to any software that is running on the computer. The ATmega16U2 microcontroller and its firmware replaces the FT232RL USB to UART bridge chip from FTDI (ftdichip.com) found on older Arduino boards (the Arduino Duemilanove www.arduino.cc/en/Main/arduinoBoardDuemilanove for example). Some clone or compatible boards still use the FTDI, or similar chip. Note that older Uno boards used the ATmega8U2 chip, but all new Arduino Uno boards use the ATmega16U2.

With the ATmega16U2 and its firmware acting as a USB to serial port bridge, the Arduino IDE can communicate with the bootloader firmware on the main microcontroller of the Arduino Uno. This allows new sketches to be loaded to the Arduino Uno. The USB to serial bridge also allows user sketches to communicate over the USB port with software running on the computer, for example with the Serial Monitor window of the Arduino IDE.

1.2.2 Bootloader

Bootloader software, or firmware, is factory loaded to the main microcontroller (part number ATmega328P) of a new Arduino Uno. When the Arduino is switched on or powered up, the first code that runs is the bootloader code. Typically, a bootloader waits a short time for a message over the serial port to see whether the user is loading a new sketch from the Arduino IDE. If a message is not received, program execution then starts at the part of memory where user sketches are loaded. In other words, it will start running whatever user sketch is found in memory. If a message is received that tells the Arduino that a sketch is ready to be uploaded from the Arduino IDE, the bootloader will receive the new sketch, byte by byte, and load it to the user sketch area of the memory. The Optiboot

bootloader software on the Arduino Uno works slightly differently to this explanation in order to start running sketches very fast, but the basic operating principle is very similar.

After a new sketch has been loaded to the Arduino from the Arduino IDE, the bootloader will start running the new sketch. From a user's point of view, the Arduino simply runs a user sketch from memory when powered up or reset, and a new sketch can be loaded to the Arduino at any time by using the Arduino IDE and USB connection. Because the USB bridge chip on the Arduino Uno has control of the reset line of the main microcontroller, the Arduino IDE can reset the microcontroller to force the bootloader to start running when a new sketch is to be loaded.

1.3 Precautions During Handling and Usage

Some precautions when handling and using an Arduino are listed below. The list does not include general workshop safety advice, but rather some things to think about before using or handling an Arduino board. For general or workshop related safety training, refer to other safety related texts which may have country or region specific advice and regulations.

1.3.1 Static Electricity

Electronic components and circuit boards, such as the Arduino Uno, can be damaged or partially damaged by static electricity, known as ESD (Electrostatic discharge). Static electricity can be stored in the human body or clothing and can discharge into a circuit board when touched, or when a person is very near to a circuit board, but not yet touching it. Static charge build-up is usually more likely to occur in dry regions and/or during dry periods of the year. Clothing made from wool, nylon or polyester can generate static electricity.

Most modern circuit boards that are susceptible to being damaged by static electricity are packaged in silver anti-static, or static shielding bags to protect them from static electricity when being shipped or stored. They usually have an ESD warning sticker attached to them. Arduino boards don't ship in anti-static bags, but are supplied in a box.

It is true that components are less susceptible to static electricity damage after they are soldered in place on a circuit board, and Arduino boards seem to be robust in this regard. The problem is that one can never easily tell what damage static electricity will do

because it is so variable in occurrence and charge strength. Some areas or regions have very low occurrences of static electricity and it may not be a problem most of the year. Users may get away with not even being aware of ESD problems in these areas. The important point is to be aware that static electricity can damage a board such as an Arduino and certain electronic components, and to take as many precautions as are feasible, depending on the individual situation and budget.

Note that not all electronic components are static sensitive. Passive parts such as capacitors, resistors and inductors are not static sensitive. Usually semiconductor devices such as microcontrollers, and other integrated circuits will be static sensitive, while parts like transistors may or may not be, depending on the device type. Refer to the datasheet from the manufacturer of the specific device for information on whether an electronic component is static sensitive or not. Many static sensitive devices have a certain amount of protection built into them to try to harden them against ESD. Again, see the datasheet for the specific device for details on built-in ESD protection.

Ideally circuit boards should be handled by persons using a grounded anti-static mat on the work surface and using a grounded anti-static wrist strap. A grounded wrist strap ensures that any static electricity in the human body is discharged to ground. A grounded mat on the work surface creates a static free environment for working with electronics. In professional environments, in addition to anti-static work surfaces and wrist straps, floors are covered with anti-static mats and personnel wear anti-static clothing and use ESD safe tools. At the very least, static electricity should be discharged from the body by touching a grounded metal surface before handling static sensitive equipment or devices.

For the hobbyist and makers on a budget, get a small bench-top anti-static mat and wrist strap for working with Arduino and electronics. Mats have studs that are used to ground them through the earth connection of a wall plug. Anti-static wrist straps connect either to ground through a wall plug or to a stud on an already grounded mat. Wrist straps have an internal 1MΩ resistor to protect the user should the ground point accidentally become live.

The list below is a summary of precaution to take when handling any static sensitive equipment.

- Avoid wearing clothing that generates static electricity
- Store static sensitive boards in silver ESD shielding bags

- Leave static sensitive components in their protective packaging until needed
- Use a grounded anti-static mat on the workbench and a grounded wrist strap
- If anti-static equipment is not available, discharge static electricity from the body on a grounded metal surface
- Avoid touching the metal parts on a circuit board – hold the board on its sides
- Be aware of environmental conditions that increase the possibility of static electricity, including low humidity air from air conditioners

1.3.2 Work Surface

Avoid metal workbenches, even if painted, as any scratches could expose the conductive metal and short circuit the Arduino board underneath. One solution to this problem is to cover the top of the workbench with an anti-static mat. Keep the work surface free of wires and small component that could easily end up under the Arduino causing a short circuit of the exposed solder joints on the bottom of the board.

1.3.3 Power

When connecting a prototype circuit, extension shield or doing any other wiring on the Arduino, make sure that the power to the Arduino is disconnected. Only power up the Arduino after finishing any wiring. Also be sure to check the wiring, especially polarity, before applying power to the system. When changing hardware connected to an Arduino, it is best to first load the default Blink sketch, or even better to load a blank sketch to overwrite the currently loaded sketch. This is because Arduino pins can individually be configured as inputs or outputs. Driving an output pin in a sketch for one hardware configuration, and then changing the hardware before loading a new sketch could result in the Arduino pin driving into an output from the new hardware, which may result in a short circuit. Loading a blank sketch is a precaution that should be taken unless you are absolutely sure that the new hardware configuration won't be damaged by the currently loaded sketch.

1.3.4 Voltage

Arduino Uno boards are rated at 5V (five volts). This voltage is the voltage that the circuitry of the Arduino operates at and the voltage levels that its output pins will drive and its input pins can handle being applied to them. Make sure that a higher voltage is not

applied to any Arduino pin and that any circuit connected to the Arduino can handle 5V – watch out for devices that can only handle lower voltages such as 3.3V. 3.3V devices can be connected to an Arduino Uno, but must use the appropriate voltage level shifters on its pins, and must use a 3.3V supply voltage, such as from the Arduino 3.3V pin. Various voltage level shifters are available, or can be built from transistors and resistors.

Although the Arduino operates at 5V, the external power supply can have a voltage of between 7V to 12V. An on-board regulator steps this voltage down to 5V.

1.3.5 Handling

Most handling precautions have to do with ESD safety as already discussed in section 1.3.1 – Static Electricity. When handling an Arduino or other circuit board it is better to pick the board up by its edges. This not only avoids touching metal parts which could be a path for ESD, but also prevents getting oils, sweat or dirt from the fingertips on the metal parts. Metal parts will eventually become dull and corroded if handled frequently. It is better to handle Arduino boards and electronic component as little as possible and avoid touching metal parts of the Arduino or electronic components. Leads of components can start to become less conductive or more resistive once they start corroding or start to get a build-up of grime. Although it is impossible to build circuits, especially breadboard circuits, without touching metal component leads at some time, the main point here is to be aware that touching them can eventually cause corrosion and dirt build-up, and to try and avoid touching them as much as possible.

1.3.6 Interfacing Precautions

Interfacing in electronics refers to the interconnecting of circuits or systems. Connecting an external circuit or component to an Arduino is an example of interfacing. When interfacing with an Arduino Uno or any other electronic system, certain precautions must be taken. Most of the precautions have to do with an understanding of basic electronics, such as not trying to draw too much current from an Arduino pin, not interfacing circuits to the Arduino that operate at too high voltages, and properly overrating components used in interface circuits. Many beginner Arduino users have not had the advantage of first getting an education in electronics, so may be at risk of damaging their Arduino or interface circuit.

One of the most important rules that beginners need to know is that when connecting an LED to an Arduino, a resistor of the appropriate resistance must always be used in series

with the LED so that the LED does not draw too much current from the Arduino pin, which can damage both the LED and the Arduino. If damage is done in this way to an Arduino Uno, it can usually be remedied by replacing the main ATmega328P microcontroller.

Unfortunately an earlier Arduino board, the Arduino NG, had a series resistor connected between a pin on the main microcontroller and Arduino digital pin 13. On this particular board, an external LED could directly be connected between pin 13 and the GND pin next to it, by inserting it directly into these two connections on the header socket of the board. The LED could then be switched on and off from a user sketch and not be damaged because of the on-board series resistor. This led to people assuming that an LED could be directly connected between pin 13 and GND of any other Arduino board. This wrong information is found on the internet and even printed in recent books. Take note that **the Arduino Uno does not have a series resistor connected to pin 13 on the board, and an LED can not safely be connected between pin 13 and GND without damaging it and/or the Arduino**.

Although this book does not teach basic electronics, it does teach interfacing principles and gives some basic interfacing examples in chapter 3. Learning electronics the hard way and burning out components and circuit boards is one way of learning. A better way is first to learn and understand how something works. New Arduino users are encouraged to learn as much about electronics and microcontrollers as possible. There are many electronics and Arduino related books and resources available to learn from.

1.4 Arduino Uno History and Revisions

Arduino Uno boards were first released in 2010 as a successor to the 2008 Arduino Duemilanove. Uno boards have the same form factor as the Duemilanove and the boards that came before the Duemilanove (Serial, NG and Diecimila). Arduino boards were invented and first manufactured in Italy, so many of the boards have Italian names. Uno means “one” in Italian and was the name given to the Arduino Uno to commemorate the release of version 1.0 of the Arduino IDE software which was also released in 2010.

The Arduino Uno has been through a few hardware revisions since its first release. An original Arduino Uno, which is now referred to as R1, REV1 or revision 1, was succeeded by a revision 2 board and then a revision 3 board. Each hardware revision added some improvements to the design of the board. At the time of writing, revision 3 is the newest

version of the Arduino Uno and has been available since 2015. The revision number is sometimes written as R3 or REV3. Currently the latest board is referred to as Rev3e in the schematic and build files and adds two components, L2 and C10, that older REV 3 boards don't have. These components form a filter on one of the main microcontroller power pins. REV3 boards have had some superficial changes since they were first released, but the revision number has not been incremented. Newer Arduino Uno REV3 boards are a teal color, instead of the blue color of older REV3 boards. Newer REV3 boards have also had some changes to the silkscreen (the white letters and markings on the circuit board), both on the top and bottom of the board, so look different from older REV3 boards. The header sockets or connectors found at the edges of Arduino Uno boards are now one-piece connectors with silkscreen labels on newer REV 3 boards. Older revision 3 boards had two separate connectors or header sockets on each side of the board with no labels on the connectors. Lastly, new Arduino Uno REV3 boards now ship with a small clear plastic base that houses the board and has additional mounting holes.

One of the more notable differences between the Arduino Uno revision 3 board and the previous two revisions is that the microcontroller chip used as a USB to serial bridge was upgraded from an ATmega8U2 to an ATmega16U2. The difference between these two chips is that the ATmega8U2 has 8K of Flash program memory and the ATmega16U2 has 16K of Flash program memory. Another difference between the R3 board and previous Arduino Uno revisions is that the number of pins on the header sockets at the edges of the board was increased, making the TWI pins also available on the other side of the board, and adding an IOREF and unused pin. Readers who would like more information, see startingelectronics.org/articles/arduino/uno-r3-r2-differences for a full list of the differences between R1, R2 and R3 Arduino Uno boards.

1.5 First Time Use and Basic Testing

Be sure to have the Arduino IDE software installed or copied to the desktop or other convenient location as described in section 1.1.5 – Programming, before starting with this section.

1.5.1 New Arduino Uno Default Behavior

Knowing the default behavior of an Arduino Uno board can help to identify if a board is working properly, or if it is faulty. This also helps with the fault finding process when dealing with a board that is suspected of being faulty.

1.5.1.1 Computer Drivers

When plugging a genuine Arduino Uno into the USB port of a computer, drivers will be loaded by the operating system of the computer. A Windows 10 computer will display a notification message indicating that drivers are being loaded the first time that a new board is connected. Subsequent times, no message is displayed. Compatible boards may need separate drivers to be downloaded and installed – check with the manufacturer.

Figure 1.10: Arduino Uno in the Windows 10 Device Manager

To check that drivers have loaded on a Windows 10 system, right click the Windows Start button, or menu icon, at the left of the bottom task bar, and then click Device Manager on the menu that pops up. This opens the Device Manager window. In the window, expand Ports (COM & LPT) by clicking the arrow at the left of it, and the Arduino should appear as a USB Serial Device with a COM port number as shown in Figure 1.10. COM port numbers found in the Device Manager may differ. They start at COM1, and can go up to COM15 and beyond.

If there is more than one device under the Ports item in Device Manager, and you are not sure which device the Arduino Uno is, leave Device Manager open and unplug the Arduino from the USB cable. The item that goes away is the Arduino. Plug the Arduino back in and it will appear back in the Device Manager window. In Figure 1.10, the Arduino Uno is the item called USB Serial Device (COM5), while COM1 above it is a different serial device.

1.5.1.2 Arduino Uno Hardware Behavior

The following behavior is expected on a new official Arduino Uno board. Bear in mind that software or firmware that controls the LEDs could be changed and updated in the future, so the LED behavior could change at some stage, however this is not highly likely.

Boards that are not official Arduino Uno boards such as clones or Arduino Uno compatible boards are expected to behave in a similar way, but may have different software loaded on them, so may not behave exactly the same way.

On LED

After plugging an Arduino Uno into an external power source, or into a computer using a USB cable, the first thing to look for is if the ON LED switches on. The ON LED indicates that the board is powered by 5 volts, either directly from the USB cable, or via the on-board regulator if external power is used.

LED Flashed by Bootloader

Immediately after applying power to an Arduino Uno, the L LED should flash or blink 3 times very fast. If this LED flashes 3 times, it indicates that the bootloader firmware is present and running. This LED will flash in the same way if the RESET button is pressed and released.

Factory or User Loaded Sketch Running

On a new official Arduino Uno board, after the bootloader has briefly flashed the L LED, the LED will start to flash or blink slowly, on for 1 second and off for 1 second. This is because the Blink sketch is factory loaded as a test and indicator to show that the Arduino is working. The Blink sketch is one of the example sketches that comes with the Arduino IDE software. Clone or compatible boards may or may not have the Blink sketch factory loaded.

The first time that a user loads a new sketch, the default Blink sketch is overwritten. When the Arduino Uno is powered up, the last user loaded sketch will run. The default Blink sketch can always be reloaded to the Arduino for testing purposes as is described in the next section.

1.5.2 Loading a Sketch to an Arduino Uno

Before loading a sketch to an Arduino Uno, start the Arduino IDE software. The correct Arduino board model and port must be selected in the IDE. A sketch can then be written,

or an example sketch opened, and loaded to the Arduino board. These steps are documented in the sections that follow.

1.5.2.1 Select Board and Port

With the Arduino IDE software started, plug the Arduino board into the USB port of the host computer. From the top menu, select Tools ► Board ► Arduino AVR Boards and then select Arduino Uno from the menu that pops out. In older versions of the IDE, the Arduino Uno is found under Tools ► Board ► Arduino/Genuino Uno.

In order to be able to load a sketch to the board, the port that the board is configured as must be selected in the IDE. From the top menu, select Tools ► Port and then the port for the Arduino board on the menu that pops up. On a Windows computer the port name will be in the form COM5 (Arduino Uno) where the number next to COM may be a different number, depending on which COM port number was assigned to the board.

Notice that after selecting the board and port, the selected board and port can be seen on the Tools menu, next to the Board: and Port: items, when it is popped out, as shown in Figure 1.11. The currently selected board and port appear in the status bar of the Arduino IDE and can be seen at the bottom right of the Arduino IDE in Figure 1.11.

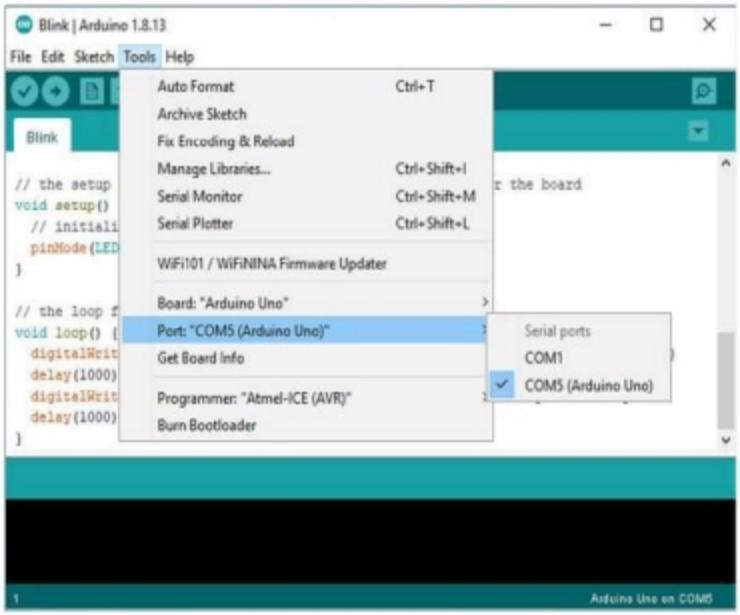


Figure 1.11: Setting up the Board and Port in the Arduino IDE

1.5.2.2 Loading a Test Sketch

Loading the Blink example sketch to the Arduino Uno is a first basic test to see if the board is working and the Arduino IDE is set up correctly. The Blink sketch comes with the Arduino IDE software and is available under the top File menu. It simply blinks or flashes the L LED on and off at a slow rate. If the Blink sketch runs, it proves that the Arduino IDE can "see" the board, that the board and port are correctly set up, and that at least some of the Arduino hardware is working.

Open the Test Sketch

Open the Blink sketch using the top menu in the Arduino IDE. The sketch is found under File ▶ Examples ▶ 01.Basics ▶ Blink and will open as text in a new Arduino IDE window when clicked.

Upload the Sketch

Once opened, the sketch can be loaded to the Arduino Uno board. Click the Upload button on the top toolbar (the second icon from the left that looks like a right pointing arrow, next to the tick mark icon), or use Sketch ▶ Upload from the top menu. If the correct Arduino board and port was selected, the sketch should compile and load to the Arduino, and then start running. The TX and RX LEDs will flash briefly during the upload.

Modify the Sketch

If the Blink sketch was already loaded to the Arduino Uno, such as with a new factory default board, no difference will be seen on the L LED when the Blink sketch is loaded again. In this case, changing the blink frequency of the LED in the sketch, and then loading the modified sketch to the Arduino, will show that a new sketch was indeed loaded properly to the Arduino, as the L LED will start blinking faster. The sketch below is a modified Blink sketch that has been changed to leave the L LED on and off for 200ms periods instead of the normal 1s periods. Start a new sketch in the IDE using File ▶ New from the top menu. Replace the code in the new IDE window with the code below.

```
void setup() {
  pinMode(LED_BUILTIN, OUTPUT);     // configure L LED pin as output
}

void loop() {
  digitalWrite(LED_BUILTIN, HIGH);  // switch L LED ON
  delay(200);                       // leave L LED ON for period
  digitalWrite(LED_BUILTIN, LOW);   // switch L LED OFF
  delay(200);                       // leave L LED OFF for period
}
```

To verify that this sketch is running on the Arduino, load it to the board and take note if the L LED is blinking faster than it does with the default Blink sketch.

1.5.2.3 Serial Port Demonstration

As already mentioned in this chapter, the USB link between the Arduino Uno and the host computer can be used by a sketch to send and receive data between the computer and Arduino. This ability can be useful when debugging a board because diagnostic information in the form of text can be sent from the Arduino and displayed on the computer. The Arduino IDE has a utility called the Serial Monitor window that can display text sent from the Arduino board. Text can also be sent to the Arduino board using the Serial Monitor window.

Serial Port Sketch Example

Open the ASCIITable example sketch from the Arduino IDE by selecting File ► Examples ► 04.Communication ► ASCIITable from the top menu. Upload this sketch to the Arduino Uno board. After uploading the sketch, nothing will be seen until the Serial Monitor window is opened.

Serial Monitor Window

In the Arduino IDE, open the Serial Monitor window by clicking the magnifying glass icon at the very right of the top toolbar, as can be seen in Figure 1.12. Alternatively, select Tools ► Serial Monitor from the top menu.

When the Serial Monitor window is opened, it connects to the Arduino from the host computer. This causes the USB bridge chip to issue a reset to the main microcontroller which restarts the sketch that is currently loaded in memory. With the ASCIITable example sketch loaded, it will start running and send text from the Arduino that will be displayed in the Serial Monitor window, as can be seen in Figure 1.12. Pressing the reset button on the Arduino Uno will also restart the loaded sketch, and text will be sent to the Serial Monitor window again. The ASCIITable sketch has been written in such a way that it sends a certain number of text characters over the USB connection, and then stops sending.

When sending text or data to the Serial Monitor window from a sketch running on the Arduino board, it is important to make sure that the baud rate set in the Serial Monitor window matches the baud rate set in the sketch. The default baud rate of the Arduino IDE Serial Monitor window is 9600 baud. This setting can be seen at the bottom of the Serial Monitor window in Figure 1.12, where it can be changed by opening the drop-down list.

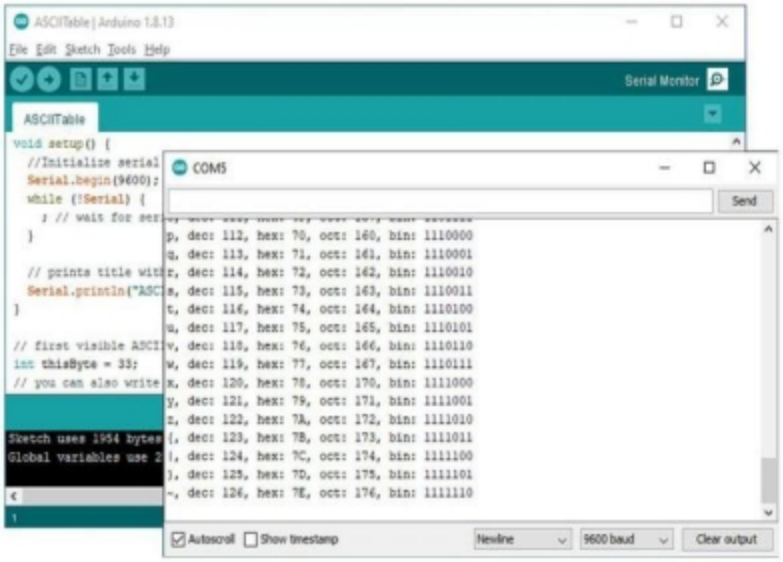


Figure 1.12: Opening the Serial Monitor Window in the Arduino IDE

It is the serial port or UART in the ATmega328P microcontroller chip that is sending and receiving serial data over the USB link via the USB bridge chip. This port is programmed using the Serial object in sketches. The baud rate of the port is set when the port is initialized at the start of the sketch with the following line of code.

```
Serial.begin(9600);
```

If the baud rate is changed here, the same baud rate must be selected in the drop-down list at the bottom of the Serial Monitor window.

In addition to being able to receive text, the Serial Monitor window can send text to the Arduino. Text typed into the text box at the top of the Serial Monitor window will be sent to the Arduino when the Send button at the right of the text box is clicked. The SerialEvent example sketch, found under File ► Examples ► 04.Communication ► SerialEvent on the top menu of the Arduino IDE, will echo back text that is sent using the Serial Monitor window. In other words, when text is typed into the top text box, and sent, it will appear in the big receive area of the Serial Monitor window because the Arduino simply sends back the text that it receives when running the SerialEvent sketch. Load the SerialEvent sketch to the Arduino, open the Serial Monitor and then send some text to test it.

1.5.3 Basic Testing

Knowing how an Arduino Uno board is supposed to behave greatly aids the testing of it. Section 1.5.1 – New Arduino Uno Default Behavior, already provides good basic information on the board that will help with fault finding, whether testing a new Arduino Uno that has just been bought to verify that it is working, or whether starting to do fault finding on a suspect or damaged board. If the board appears to be starting up correctly, and is recognized by the host computer (drivers are loaded and it can be seen in the operating system, for example in the Device Manager of Windows computers), the next step is to try to load a sketch as described in section 1.5.2 of this chapter – Loading a Sketch to an Arduino Uno. The Blink sketch is a good sketch to use for very basic testing.

If an Arduino board that was known to be working becomes faulty, use the steps that follow to do some basic initial fault finding. More advanced fault finding requires a better understanding of the hardware which is explained in later chapters of this book.

1.5.3.1 Visually Inspect the Board

Check the board for mechanical damage such as broken tracks on the circuit board or obviously damaged components or bent connectors with pins that may be shorted together (ICSP headers). Also look for burned out components, which may contain burn marks or show burn marks on the circuit board around them. Don't power up the board if it shows obvious signs of damage. First repair the board or replace damaged components before powering up the board. It is better to first power a newly repaired board from external power to avoid the risk of damaging the host computer should anything go wrong.

1.5.3.2 Power LED and Voltages

If the board appears to be physically undamaged, power up the board and check that the ON LED lights up. The ON LED indicates that 5V is present on the board. If the LED does not light up, use a multimeter set to the DC voltage scale to check if 5V is present between the GND and 5V pins. If 5V is present, it could indicate a blown ON LED.

Check that the voltage on the 5V pin is actually 5V or a value near to it. Low voltage from a USB connection could be a possible problem. If 5V is not present when the board is powered from an external power supply, the on-board regulator could be faulty. Also check that the external voltage can be measured on the Vin pin to make sure that the external supply voltage is getting to the board.

1.5.3.3 Check Expected Default Behavior

See if the board shows the expected default behavior when powered up. If the L LED does not blink 3 times rapidly at power up, this could indicate that the main microcontroller is damaged. If the main microcontroller was replaced, it may not have a bootloader present, or may have been shipped with a different bootloader. A new microcontroller needs its internal configuration fuses set to the values required by the Arduino Uno.

1.5.3.4 Is the Board Recognized by the Host Computer?

Make sure that the board appears in the Port selection list of the Arduino IDE under the Tools menu, or that it can be found in the operating system, such as in the Device Manager on a Windows system. If the board is not found, it could indicate a problem with the USB bridge chip on the Arduino Uno, or a driver problem on the host computer.

On a Windows system the COM port number changes when changing boards – that is, if you have more than one Arduino Uno and swap them on the computer's USB port. In this case a sketch won't upload to a new board if the old board was unplugged and a new or different board plugged into the USB port of the host computer, because the new port must first be selected in the Arduino IDE. Select the new COM port from the Port menu found under the top Tools menu before attempting to upload a sketch to the new board.

1.5.3.5 Load a Test Sketch

If all of the tests in the previous sections pass, then the final test is to load a sketch to the Arduino to make sure that it uploads correctly and starts running. As already mentioned, the Blink example sketch is a good sketch to use for basic testing. If the Blink sketch does not work then try one of the serial port tests to see if the Arduino can send back text over the USB link to the Serial Monitor window, as described in section 1.5.2.3 of this chapter – Serial Port Demonstration.

1.6 Arduino Uno References and Help

Several resources are available on the Arduino website to help new users get started with Arduino, as well as any Arduino users to get support. This section is mostly for the benefit of new Arduino users, to help them find all the basic information needed to get started with Arduino boards. Find the supporting website for this book at wspublishing.net and click the book name to go to its area on the website. Also find resources for this book on the Elektor website at www.elektor.com where the book can be searched for.

1.6.1 Installing Software

Help documentation on installing the Arduino IDE software is available for Windows, MAC OS X and Linux. Follow the links below for your operating system.

1.6.1.1 Windows

www.arduino.cc/en/Guide/Windows

1.6.1.2 MAC OS X

www.arduino.cc/en/Guide/MacOSX

1.6.1.3 Linux

www.arduino.cc/en/Guide/Linux

1.6.2 Getting Started, Examples and Reference

Getting started guides, examples, and a software reference are all available on the Arduino website. Getting started guides and examples are a great way for new Arduino users to learn about building projects with Arduino and learning to write sketches.

1.6.2.1 Getting Started Guides

Links to getting started guides for any Arduino board can be found at:
www.arduino.cc/en/Guide

Arduino Uno specific getting started guide:
www.arduino.cc/en/Guide/ArduinoUno

1.6.2.2 Arduino Examples and Tutorials

Various tutorial categories can be found at www.arduino.cc/en/Tutorial/HomePage on the Arduino website.

1.6.2.3 Building Breadboard Circuits

A tutorial series at startingelectronics.org/beginners/start-electronics-now shows how to build breadboard circuits with some Arduino examples.

1.6.2.4 Arduino Software Reference

Arduino language and library reference: www.arduino.cc/reference/en/

1.6.3 Getting Help

Find the **Arduino Help Center** at support.arduino.cc/hc/en-us which contains a searchable help index, as well as several categories of FAQs (Frequently Asked Questions).

The **Arduino Playground** at playground.arduino.cc/Main/ElectroInfoResources is a user contributed collection of documentation and other useful information.

The **Arduino Forum** at forum.arduino.cc is a place where anyone can post a question that will be answered by community members. The forum can also be searched for other user's questions which may already have answers. It is also a place where more experienced Arduino users can help others by answering questions.

1.6.4 Related Open-Source Projects

Several open-source projects are related to Arduino, are used in education like Arduino, and are useful to Arduino users. An overview of Arduino would not be complete without mentioning the projects in the sections that follow.

1.6.4.1 Fritzing

Fritzing, found at fritzing.org on the web, provide a software package that can be used to draw Arduino breadboard circuits and then turn them into custom made printed circuit boards or PCBs. Like the Arduino IDE software, it is available for Windows, Mac OS X and Linux. It is an easier way for beginners to create an Arduino shield or custom PCB without the need to learn a full-blown EDA CAD package. Fritzing can also be used to draw Arduino breadboard circuits for tutorials or other documentation.

The software has a drag-and-drop type of interface where common electronic components can be placed on a breadboard visually, and wired up to an Arduino Uno or other board. In the Fritzing software there are three different views of the same circuit. A breadboard view shows the breadboard circuit as it is being built. A schematic or circuit diagram view of the same breadboard circuit can be seen under the schematic tab in the software. Under the PCB tab is a view of the shield PCB and the components placed on the breadboard. The PCB can be built here by moving the components to the desired areas on the board and then joining them using tracks that will be the copper tracks of the PCB when it is manufactured. When laying out the PCB, lines show the electrical connections made on the breadboard so that joining the components correctly using PCB tracks is made easy.

1.6.4.2 Wiring

Wiring is the project that Arduino hardware and the Arduino IDE software is largely based on. Find the Wiring project at wiring.org.co on the web. Wiring was already mentioned in this chapter in section 1.1.3 – Uses of the Arduino Uno. The Arduino Uno IDE is based on the Wiring IDE which in turn is based on the Processing IDE. Processing is described in the next section.

1.6.4.3 Processing

Processing, found at processing.org on the web, is a programming language based on Java, and a software package for developing applications on a computer. It was developed for use in the visual arts. Like the Arduino IDE, it uses the concept of sketches and looks very similar to the Arduino IDE software. As with Arduino it is available cross-platform for Windows, Mac OS X and Linux.

Processing is often used in conjunction with Arduino where a custom application is needed on a computer that communicates with an Arduino board via the USB cable. Some of the Arduino IDE example programs communicate with a Processing application. One such example is PhysicalPixel, found in the Arduino IDE under File ► Examples ► 04.Communication ► PhysicalPixel from the top Arduino IDE menu. Code at the bottom of this example sketch is commented out – meaning that it is made into one big comment so that the Arduino IDE and compile tools won't try to interpret or compile it. The idea is to copy this code from the bottom of the Arduino sketch and paste it into the Processing IDE, where it will be uncommented. Then load the Arduino sketch to the Arduino, and run the Processing sketch on the computer. When the Processing sketch runs, it displays a window with a small square in the middle of it. Moving the mouse cursor over the square switches the Arduino Uno L LED on. Moving the mouse cursor off the square switches the Arduino L LED off. This is done by the Processing sketch communicating with the Arduino sketch over the serial USB link – similar to using the Serial Monitor window. For more information, see www.arduino.cc/en/Tutorial/BuiltInExamples/PhysicalPixel on the Arduino website.

1.6.5 Arduino Uno Boards on the Web

Links to the web pages on the Arduino website for the Arduino Uno and Arduino Uno SMD boards follow.

1.6.5.1 Arduino Uno Web Page

Find the Arduino Uno web page at store.arduino.cc/arduino-uno-rev3 which includes basic technical information and links to the circuit diagram and board build files in electronic format.

1.6.5.2 Arduino Uno SMD Web Page

Go to store.arduino.cc/arduino-uno-rev3-smd for the newest information on the Arduino Uno SMD, or to www.arduino.cc/en/Main/ArduinoBoardUnoSMD for the Arduino Uno SMD board page on the Arduino website. An electronic copy of the circuit diagram for the Arduino Uno SMD can be found here as well as the build files.

Chapter 2 • Hardware Technical Information

A description of the Arduino Uno hardware technical information is contained in this chapter, split up into various categories. This chapter and chapter 1 are the hardware user manual chapters of this book.

Whereas chapter 1 gave a broad overview of the Arduino Uno hardware, this chapter gives more specific technical information on the hardware. It "zooms in" to the details of the Arduino Uno and gives more information on the main microcontroller, memory types and sizes, power supply technical details, technical specifications on the programming and user headers, and other hardware on the board.

In this Chapter

- Arduino Uno microcontroller and memory
- External and USB power specifications and connection
- Board operating frequency
- LED indicator and reset button information
- Header sockets pinout and detailed pin functions
- Programming headers pinout and functions

2.1 Microcontroller

The main microcontroller on an Arduino Uno is an ATmega328P-PU 8-bit AVR from Microchip. On the Arduino Uno SMD, the main microcontroller has a part number of ATmega328P-MU. Essentially they are the same device from the same microcontroller family, but the last two digits of the part number differ as they are manufactured in different physical housings or packages. Table 2.1 shows the Arduino Uno boards with order codes, microcontroller part number, and microcontroller chip packages.

Table 2.1: Arduino Uno Boards and Microcontroller Part Numbers

Board	Board Code	Microcontroller	Chip Package
Arduino Uno	A000066	ATmega328P-PU	PDIP
Arduino Uno SMD	A000073	ATmega328P-MU	QFN

On the Arduino Uno, the ATmega328P-PU is housed in a 28-pin Plastic Dual Inline Package (PDIP) which can be plugged into and removed from the socket on the board. Figure 2.1 shows two ATmega328P-PU microcontroller chips that can be used to replace the same device on an Arduino Uno board, while the Arduino Uno SMD has its surface mounting ATmega328P-MU housed in a Quad Flat No-Lead (QFN) package.

Figure 2.1: Two ATmega328P-PU Microcontroller Chips in 28 Pin PDIP Housings

Some clone or Arduino Uno compatible boards use yet a different microcontroller package, although still the same ATmega328P device. The microcontroller on the Arduino Uno compatible board in Figure 1.9 of Chapter 1 is housed in a Thin-plastic Quad Flat Package (TQFP). It is bigger than the QFN package, but still surface mount soldered to the top of the board, so is not easily replaceable like a socketed chip.

2.2 Atmel, Microchip and AVR

Originally Atmel was the designer and manufacturer of AVR microcontrollers including the ATmega328P found on Arduino Uno boards. In 2016, Microchip (www.microchip.com) bought Atmel and is now the owner and manufacturer of AVR microcontrollers. At the time of writing, even new Arduino Uno boards have AVR microcontrollers that still have the Atmel name on them. Many older books and internet sources still make reference to Atmel as the AVR microcontroller manufacturer, and still link to the old Atmel website which redirects to the Microchip website. Note that AVR is the microcontroller family of which the ATmega328P and ATmega16U2 (USB bridge chip on the Arduino Uno) are members. AVR refers to the internal processor architecture of these chips, which is 8-bit RISC (Reduced Instruction Set Computer).

2.3 Memory

All memory on the Arduino Uno board is found in the ATmega328P microcontroller chip and is summarized in table 2.2.

Table 2.2: Arduino Uno (ATmega328P) Memory Sizes

Memory Type:	Flash (Program)	SRAM (Data)	EEPROM (Data)
Size in Bytes:	32K (32,768 bytes) 31.5K available by default because of bootloader using a 512 byte segment	2K (2,048 bytes)	1K (1,024 bytes)

2.3.1 Flash Memory

Flash memory is non-volatile memory used to store the program instructions of an Arduino sketch or program. Non-volatile memory is memory that retains its contents even if the power is switched off. When an Arduino sketch is loaded to an Arduino, it is loaded to the Flash memory of the ATmega328P microcontroller chip. Once a sketch is loaded to the Flash memory, every time the power to the Arduino is switched on, the program will start running.

2.3.1.1 Flash Memory Size

The ATmega328P microcontroller, and therefore the Arduino Uno, has 32K bytes (32 × 1024 bytes) of Flash memory, where a byte is 8 bits wide and 1K = 1024 bytes. Because the bootloader software uses up 512 bytes, or 0.5K, of the Flash memory, 31.5K is available for user programs or sketches. Although the bootloader size is not exactly 512 bytes, a 512 byte sector of Flash is reserved for it.

2.3.1.2 Flash Wear

All Flash memory is subject to wear and has a certain lifetime determined by how many times it is erased and written to or programmed. The life of the Flash program memory of the ATmega328P due to Flash wear is 10,000 Write/Erase cycles.

2.3.1.3 Data Retention

Data retention, or how long a program remains in the Flash memory before failure, is less than 1 PPM (Part Per Million) over 20 years at a temperature of 85 degrees Celsius, or 100 years at a temperature of 25 degrees Celsius.

2.3.2 SRAM

SRAM (Static Random Access Memory) is volatile memory that is used to store variables or data from an Arduino sketch or program. Part of the SRAM is used for special memory areas called the heap and the stack. Users or programmers need not worry about allocating memory as this is done automatically by the programming tools that are run when loading a sketch to the Arduino using the Arduino IDE.

2.3.2.1 Volatile Memory

Volatile memory is memory that loses its contents when its power is switched off. Part of the SRAM is initialized with data when the program in the main Flash memory starts running, depending on which data is allocated to this memory, for example global variables. Other parts of the SRAM are initialized with values while the sketch is running. All of these values are lost from SRAM memory when the Arduino is powered off.

2.3.2.2 SRAM Size

The ATmega328P microcontroller, and therefore the Arduino Uno, has 2K bytes (2 × 1024 bytes) of SRAM, where a byte is 8 bits wide and 1K = 1024 bytes.

2.3.3 EEPROM

EEPROM (Electrically Erasable Programmable Read Only Memory) is non-volatile memory used for data storage. Data stored in EEPROM will not be lost if power to the Arduino is switched off.

2.3.3.1 EEPROM Programming

EEPROM is user programmable and is supported in the Arduino IDE by the EEPROM library, which includes such functions as *EEPROM.write()* and *EEPROM.read()* for writing data to, and reading data from the EEPROM. Refer to the library reference www.arduino.cc/en/Reference/EEPROM for more information on using the EEPROM in sketches. EEPROM example sketches can be found in the Arduino IDE from the top menu under File ►Examples ►EEPROM where EEPROM is found near the bottom of the Examples menu.

2.3.3.2 EEPROM Size

The ATmega328P microcontroller, and therefore the Arduino Uno, has 1K bytes (1 × 1024 bytes) of EEPROM, where a byte is 8 bits wide and 1K = 1024 bytes.

2.3.3.3 EEPROM Wear

As with Flash program memory on the ATmega328P microcontroller, EEPROM is subject to wear as it is erased and programmed. EEPROM on the ATmega328P has a lifetime of 100,000 Write/Erase cycles.

2.3.4 Adding External Memory

External data memory can be added to an Arduino Uno in the form of Flash chips, EEPROM chips, or SD cards. Although there are no options to directly add external memory to an Arduino Uno, external memory can be added using the header sockets of the board. Connection of external memory can be in the form of a shield, or breadboard circuit. Ready made shields such as data logging shields usually have an SD card or micro SD card socket. Some shields are combination shields, such as most Ethernet shields that have both an Ethernet connection and micro SD card socket.

2.3.4.1 SD Cards (SPI Interface)

SD cards or micro SD cards can be connected to an Arduino Uno using a shield, such as mentioned above, or with an SD card socket breakout board, or module, with level shifters as shown in Figure 2.2. SD cards operate at 3.3V and are not 5V compatible, so it is

important to make sure that the shield or SD card module used with an Arduino Uno is specified to work with 5V Arduino boards. The SD card module shown in Figure 2.2 has a 3.3V regulator and a level shifting circuit for correct interfacing between a 5V Arduino and 3.3V SD card. The module is supplied with 5V from the Arduino which is stepped down to 3.3V by the regulator on the module, and then fed to the SD card. Although the voltage of an SD card is given as 3.3V, most SD cards operate over a range of voltages, usually between 2.7V and 3.6V.

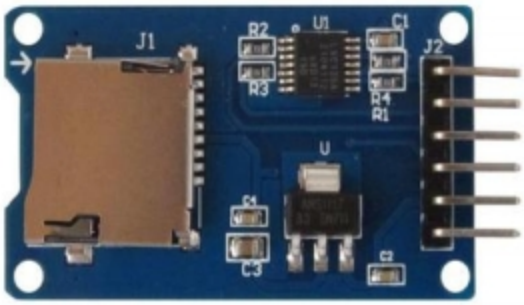


Figure 2.2: SD Card Module

SD cards have two operating modes, namely SD mode and SPI mode. SD cards will always use SPI mode when connected to an Arduino Uno. SD mode is patented and requires licensing for any equipment that uses it. SPI mode has no restrictions and can be used freely. To get the fastest data transfers from a standard SD card, SD mode would need to be used because it can transfer 4 bits in parallel at a time. An Arduino Uno is not fast enough to get the full data transfer speed from an SD card, even if it were able to use the SD bus mode.

See www.arduino.cc/en/Reference/SD for more information on the Arduino SD card library. SD card examples can be found in the Arduino IDE from the top menu under File ► Examples ► SD where SD is below the middle of the Examples menu. The CardInfo sketch on the SD examples menu is a good sketch to use for testing an SD card and to check if an SD card is correctly wired to an Arduino. SD cards interface or connect to the Arduino Uno using the SPI (Serial Peripheral Interface) pins of the board. See section 3.1.2 – Shared SPI Pins, in Chapter 3 for more on the location of the SPI pins on the Arduino Uno. Section 3.3.5 – SPI Bus Pins, contains more information and an interfacing example on how to connect SD card modules to an Arduino Uno.

2.3.4.2 Flash and EEPROM Chips (SPI / TWI Interface)

Flash or EEPROM chips that use either the SPI (Serial Peripheral Interface) or TWI (Two Wire Interface) can be connected to the appropriate pins of an Arduino Uno. See sections 3.1.1 and 3.1.2 in the next chapter for TWI and SPI pinouts on the Arduino Uno. Section 3.3.4 contains a TWI interfacing example and section 3.3.5 contains a SPI interfacing example.

Both SPI and TWI are serial interface bus protocols that have software libraries available in the Arduino IDE. TWI is compatible with the I²C (Inter-Integrated Circuit) serial protocol, so I²C memory chips will also work with the Arduino Uno. I²C is pronounced "I squared C" (eye squared see) and may also be written as IIC or I2C.

The ATmega328P microcontroller has built-in hardware that handles SPI and TWI serial communications. For this reason, SPI and TWI devices can only be connected to certain pins of the Arduino Uno, although an external library could be used to emulate SPI and TWI on other pins. SPI and TWI are not limited to only adding external memory to the Arduino Uno. Other devices that use SPI or TWI interfaces can also be connected to an Arduino, such as RTC (Real Time Clock) chips and I/O (Input/Output) expansion chips.

2.3.4.3 SPI Devices

SPI devices are supported in the Arduino IDE by the SPI library. See the library reference www.arduino.cc/en/Reference/SPI for more information on the SPI library. SPI examples can be found in the Arduino IDE from the top menu under File ► Examples ► SPI where SPI is found near the bottom of the Examples menu.

2.3.4.4 TWI and I²C Devices

TWI and I²C devices are supported in the Arduino IDE by the Wire library. See the library reference www.arduino.cc/en/Reference/Wire for more information on the Wire library. Find examples that use the Wire library in the Arduino IDE from the top menu under File ► Examples ► Wire where Wire is found at the bottom of the Examples menu.

2.4 Power and USB

An external DC power supply, battery or USB connection can be used to provide power to an Arduino Uno, as explained in the sections that follow.

2.4.1 USB Power

Power can be supplied to the Arduino Uno from a computer via the USB connection, which supplies 5V. This is convenient for prototype and experimental use, but especially for a classroom environment where eliminating the need for an external power supply can save on costs.

A USB 2.0 computer port is specified to provide 5V ± 5% and is able to deliver up to 500mA of current. Arduino Uno boards have a 500mA resettable fuse protecting the 5V supply from the USB port. A resettable fuse will break the circuit if current more than its rated current is drawn by an overload or short circuit. The fuse will automatically reset after the overload or short circuit is removed. Bear in mind that a small amount of the USB 5V voltage will be dropped across the fuse. This voltage will increase as more current is drawn through the fuse, which means that as this voltage drop increases, the voltage measured on the Arduino 5V pin will decrease by the same amount.

2.4.2 USB Connection and Cable

An Arduino Uno is a USB 2.0 full-speed device, which has a data throughput of 12Mbit/s and can have a USB cable length of up to 5m (16.4 feet), according to the USB 2.0 specification.

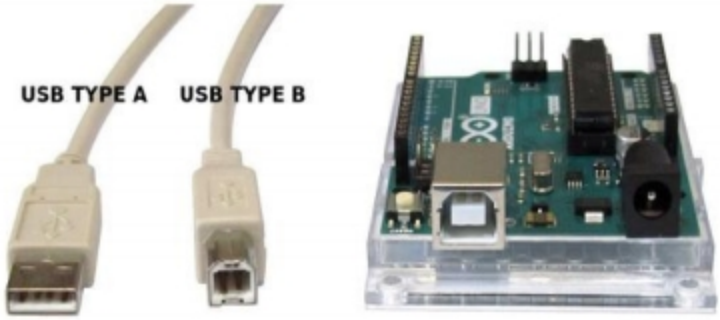


Figure 2.3: USB Type A-B Cable and Arduino Uno Showing B Type Connector

In addition to being able to power the Arduino Uno, the USB connection also allows the Arduino to be programmed from the Arduino IDE. An ordinary USB printer cable is used to connect the Arduino Uno to a computer. This type of cable uses USB connectors of type A

on the computer side and type B on the Arduino side. These cables are referred to as A-B USB cables. Figure 2.3 shows the type A and type B connectors of a USB cable and the end of an Arduino Uno board showing the type B connector and external power jack.

2.4.3 External Power

External power can be supplied via the barrel connector jack on the Arduino Uno, allowing the board to run standalone without the need for a computer USB connection as shown in Figure 2.4.

Figure 2.4: Arduino Uno Powered by External DC Power Supply
Inset: Barrel Connector

An external DC (Direct Current) power supply with a voltage of 7V to 12V, capable of delivering at least 500mA is recommended. The center pin of the 2.1mm barrel connector must be connected to the positive connection from the external power supply. Actual current requirements for the Arduino depend on the project that it is used in. Current drawn will be a small amount from the Arduino itself and the remainder will be from the circuit or shield that is connected to it. Refer to chapter 4, section 4.1.4, for the pinout of the external power jack.

If the USB cable is plugged into the Arduino Uno for programming, and an external power supply is connected, the Arduino will automatically be powered from the external power supply. Power supply circuitry detects when external power is connected and automatically switches to it if found.

2.4.4 Battery Power

A battery can be connected to either the external 2.1mm power jack or through the Vin pin of the Arduino Uno. Whenever connecting a battery, be sure to check the polarity of the connection – connect the center pin of the barrel power jack to positive of the battery and negative of the battery to GND. If connecting the battery to the Vin pin on the Arduino Uno header socket, connect positive from the battery to the Vin pin and negative to a GND pin.

Figure 2.5: Battery Holder, Rechargeable Battery, 9V Battery and Battery Clips

Common battery voltages that are safe to use with an Arduino are 9V and 12V. Small 9V batteries of the PP3 or E-block size are not practical for powering an Arduino Uno for long periods of time. They may be used for short periods, such as for a battery controlled robot car that is only operated intermittently. Other options are to use a battery holder that holds

six 1.5V AA cells, making up 9V. This will be around 7.2V if rechargeable cells are used which usually supply 1.2V each. These holders have PP3 type battery terminals that connect to a PP3 type battery connector or clip. Rechargeable PP3 or E-block batteries have a voltage of 8.4V because internally they consist of seven 1.2V cells. Figure 2.5 shows a 6 cell holder, 8.4V rechargeable battery, 9V PP3 battery and two battery clips. The battery clip at the top of the image has a barrel connector for connecting to the external power jack of an Arduino Uno. These battery cables can be bought ready assembled, or built from parts bought separately. The red wire on a battery clip is the positive lead and the black wire is the negative lead. Bigger 9V batteries are available, such as PP9. Sealed 12V gel type batteries that are commonly used in house alarm systems are available in several sizes and power ratings.

Consider storing batteries in plastic zip-lock bags to prevent the terminals from being shorted together by metal parts – don't just throw the battery into a spares box that may have other components in it that could short the battery.

Figure 2.6 shows the difference between connecting a battery to the external jack as opposed to using the Vin pin found on the POWER section of the header socket of the Arduino Uno. Vin is clearly labeled on the top of the Arduino Uno board as well as silkscreened on both sides of the header socket on newer Arduino Uno R3 boards.

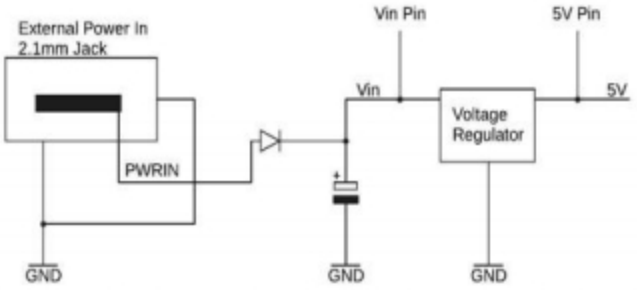


Figure 2.6: External Power In and Vin Circuit of an Arduino Uno

As can be seen in the figure, the positive connection from the 2.1mm jack (center pin) connects to the Vin pin via a diode. The diode offers some crude reverse polarity protection on the input jack, but does not protect the Vin input. If power is connected to the external jack, the voltage seen on the Vin pin will be the input voltage minus the voltage drop across the diode.

2.4.5 Operating Voltage

Because the internal circuitry and main microcontroller operate at a voltage of 5V, the Arduino Uno is considered to be a 5V device. Input/output (I/O) or digital pins operate at 5V levels because the ATmega328P is supplied with 5V from either USB or from an external power supply, where the on-board voltage regulator drops the external power supply voltage to 5V. External devices connected to an Arduino Uno must be rated at 5V, or have the necessary voltage regulator and level shifters for the pins to avoid damaging them. Watch out for some devices that may be rated at 3.3V. They may be available as a module or on a shield with the necessary regulator and level shifters built in. If not, they can still be interfaced to an Arduino Uno, but only through the correct voltage level adjusting circuit.

2.5 Operating Frequency

The main microcontroller (ATmega328P) of the Arduino Uno runs at a frequency of 16MHz. A 16MHz ceramic resonator is connected to the oscillator pins of the ATmega328P, providing a 16MHz square wave clock pulse. A ceramic resonator is not as accurate as a quartz crystal.

A 16MHz crystal connected to the ATmega16U2 USB to serial bridge microcontroller allows this device to run at frequency of 16MHz.

2.6 LED Indicators and Reset Button

Four LEDs are found on the Arduino Uno as summarized in Table 2.3. The LEDs are clearly marked on the top of the Arduino Uno board and the same LED names are used in the left column of the table, as on the board. One reset button can be found next to the USB connector. The LEDs and reset button are described further in the sections below.

Arduino Uno clone or compatible boards may use different colored LEDs than official Arduino boards. These boards may have a different layout as well, which means that the LEDs may be found in different locations on the boards when compared to official boards. Usually the LED names on clone boards are the same as on official boards.

Table 2.3: Arduino Uno LED Names and Functions

LED Name	Purpose	Description
ON	Shows that the Arduino Uno is powered and on.	This LED switches on if the Arduino is powered from USB or an external power supply. It indicates that the Arduino is getting 5V from USB or from an external power supply via the on-board 5V regulator.
TX	Shows Arduino is transmitting data.	Arduino is transmitting data over the serial UART to USB link.
RX	Shows Arduino is receiving data.	Arduino is receiving data over the serial UART to USB link.
L	User programmable. Used by bootloader.	User programmable LED connected to pin 13 of the Arduino Uno. Blinked fast 3 times by the bootloader to indicate that the bootloader code started running.

2.6.1 ON LED

The green ON LED will light up if 5V is present on the board when the board is externally or USB powered.

2.6.2 L LED

The yellow L LED is connected to digital pin 13 of the Arduino Uno. When the Arduino Uno is powered up, the bootloader starts running and flashes or blinks the L LED fast 3 times. After the bootloader has handed control over to the currently loaded sketch, the L LED is free to be used by the sketch. It can be accessed in a sketch by setting pin 13 as an output and then using the digitalWrite() function to either set it HIGH (switches the LED on) or LOW (switches the LED off). The L LED can also be accessed in sketches by referring to it as LED_BUILTIN. Open the Blink sketch in the Arduino IDE from the top menu under File ► Examples ► 01.Basics ► Blink for an example of how to program this LED.

The L LED is buffered through a unity gain op-amp circuit in revision 3 Arduino Uno boards, so switching this LED on will draw virtually no current from Arduino Uno pin 13. Pin 13 just supplies a signal to the op-amp, and current is supplied to the LED by the op-amp (only on revision 3 boards).

2.6.3 TX LED

TX = Transmit. The yellow TX LED will light up and flash or blink whenever data is sent or transmitted from the ATmega328P serial or UART transmit line to the USB bridge chip. The TX LED lights up whether data being transmitted is from the bootloader or a user sketch. If data is transmitted continually, this LED will stay on rather than blink.

2.6.4 RX LED

RX = Receive. The yellow RX LED will light up and flash or blink whenever data is received by the ATmega328P from the USB bridge chip. The RX LED lights up when data is received by the bootloader on the ATmega328P from the Arduino IDE, when loading a new sketch. It will also light up if data is being sent over the USB to serial link from the computer and received by a user sketch. It will stay on if data is received continually.

2.6.5 Reset Button

Pressing the reset button next to the USB connector on an Arduino Uno R3 board issues a manual reset to the ATmega328P microcontroller which causes it to set internal registers to their default states and restart the microcontroller. This means that the bootloader will restart and then the sketch that is in memory will start running from the beginning. On Arduino Uno boards before revision 3, the reset button was positioned near the ICSP header at the opposite end of the board from the USB connector.

2.7 User Pin Headers

User pin headers are found on each side of the Arduino Uno down the length of the Arduino. Although these “pins” are actually sockets, they are usually referred to as pins in the Arduino documentation. They provide power and ground connections as well as access to the digital and analog pins of the ATmega328P microcontroller. These header sockets allow external hardware to be connected to the Arduino in the form of shields, custom circuit boards or prototyping boards such as electronic breadboards. Each pin type is discussed in further detail in the sections that follow.

2.7.1 Power Pins

5V (five volts), 3.3V (three point three volts), Vin (voltage in) and GND (ground) are the four types of power pins found on the Arduino Uno.

2.7.1.1 GND Pins

GND pins are connected to 0V or negative from the power supply. All voltages are measured relative GND. In other words the negative lead of a voltmeter, multimeter set to voltage, or oscilloscope connects to GND when making voltage measurements.

2.7.1.2 5V Pin

The 5V pin gets its voltage from either the USB, or from the on-board 5V regulator if the board is powered from an external power supply. 5V can be used by a shield or other circuit that is interfaced to the Arduino Uno.

2.7.1.3 3.3V Pin

3.3V on the Arduino Uno is derived from 5V through a 3.3V regulator. The 3.3V regulator is able to supply a maximum of 150mA of current. Older Arduino boards were able to supply only 50mA on the 3.3V pin which was derived from a 3.3V output pin on the FT232RL chip. This chip is replaced by an ATmega16U2 and separate 3.3V regulator on an Arduino Uno.

2.7.1.4 Vin Pin

Vin is the voltage from an external power supply connected to the barrel connector, but after it has been fed through a protection diode (see Figure 2.6), so the voltage of Vin will be slightly lower than the actual input voltage from the power supply. Subtract around 0.7V to 0.8V from the input voltage to get the value of Vin. This is because the diode has approximately 0.7 to 0.8V forward voltage drop. Refer to section 2.4.4 for more information on battery powering the Arduino through this pin. Vin is available to any shield plugged in.

2.7.2 IOREF Pin and Unconnected Pin

The IOREF pin was added to revision 3 Arduino Uno boards. IOREF is 5V on an Arduino Uno – it simply connects to the on-board 5V supply. The idea of this pin is that it is set to the voltage that the Arduino operates at. This means that it is 3.3V on Arduino boards that operate at 3.3V and 5V on Arduino boards that operate at 5V. It is intended as a reference voltage for shields so that they can determine whether they are connected to a 3.3V or 5V Arduino. This allows the shield to enable voltage level shifters or translators if necessary.

Clone or compatible Arduino boards may not have this pin available. It is not present on Arduino Uno R1 and Arduino Uno R2 boards.

Immediately next to the IOREF pin is an unused pin that is unconnected and reserved for future purposes. The Arduino header that the IOREF and unused pin are connected to increased pin count from 6 pins to 8 pins in the Arduino Uno R3 as a result of these two pins being added to revision 3 boards. Revision 1 and 2 boards have a 6-pin header in this position on the board.

2.7.3 RESET Pin

The Arduino reset pin connects to the ATmega328P microcontroller reset pin. This pin is active low and enables shields to have a reset button placed on them that can reset the main microcontroller. In cases where a shield blocks access to the reset button on the Arduino Uno, a reset button on the shield enables easy manual resetting of the Arduino. Bringing the voltage level on this pin LOW (to GND) resets the main microcontroller. It does not reset the ATmega16U2 USB to serial bridge microcontroller.

2.7.4 Digital and PWM Pins

Digital pins are clearly marked as such on the Arduino Uno board and are numbered from 0 to 13 which means that there are 14 digital pins – 1 to 13 would be 13 pins, but because the pins are numbered starting from 0, 0 to 13 means there are 14 pins. Digital pins are highly configurable and can individually be set up as input pins or output pins from within a sketch. Some of the pins have special functions available on them and can be set up as PWM (Pulse Width Modulation) pins. Pins 0 and 1 are serial port or UART pins which can instead be configured as input or output pins. Each digital pin configuration is described in more details in the sections that follow.

2.7.4.1 Output Pins

When a digital pin is configured as an output pin, it can drive a load such as an LED with a series resistor or other low current devices such as some buzzers. Driving a load means switching power to the load on. A load can be connected in either a current sourcing or current sinking configuration as described in section 3.3.1.1, under the Current Sourcing and Current Sinking headings, in the next chapter. To avoid destroying, or partially destroying the microcontroller, it is important not to connect a load to any of the digital pins that will draw more current from the pin than the pin can safely deliver. More on this in the next section.

To set up a digital pin as an output, call the pinMode() function in the setup() part of the sketch as shown in the code listing below. To switch the digital pin that is configured as an output on or high, which will make the voltage of the pin change to 5V, call the function digitalWrite() with the second argument set to HIGH. To switch the voltage level on the pin to low, also called GND, 0V or off, call digitalWrite() with the second argument set to LOW as shown in the sketch below. Whether the load on the pin switches on or off depends on whether it is wired in a current sourcing or current sinking configuration.

```
void setup() {
  pinMode(12, OUTPUT);    // configure digital pin 12 as an output pin
}

void loop() {
  digitalWrite(12, HIGH); // set pin 12 voltage to 5V on an Arduino Uno
  delay(500);
  digitalWrite(12, LOW);  // set pin 12 voltage to 0V or GND
  delay(500);
}
```

The above sketch is a simple variation on the Blink example sketch that has already been described in Chapter 1. This sketch configures digital pin 12 as an output pin instead of pin 13 that the Blink sketch does. It then drives pin 12 high and low which will switch an LED on and off if it is attached to pin 12 using a series resistor. In the Blink sketch, digital pin 13 for an Arduino Uno is defined as LED_BUILTIN. LED_BUILTIN is just a name that is defined for the on-board L LED on Arduino boards. On other Arduino boards the L LED may be connected to a pin other than pin 13. LED_BUILTIN will then be defined as the correct pin for the L LED on that board. In other words, LED_BUILTIN is defined as the L LED pin for each different Arduino board so that any sketch that uses the L LED will work on any Arduino board, irrespective of which pin the L LED is connected to, so long as the L LED is referred to as LED_BUILTIN in the sketch.

LED_BUILTIN is defined within source code in the Arduino IDE in a similar way that is described here, which is illustrated with an example sketch that follows. The correct source code file with the correct definition, or pin number, for LED_BUILTIN is linked to an Arduino sketch project for the board that is selected in the Arduino IDE. The sketch that follows is an alternative to the above sketch that gives the digital pin a user friendly name, instead of referring to it by its pin number – this is how LED_BUILTIN is defined in source code included with the Arduino IDE. The advantage of this method is that if the pin number

needs to be changed, it can be changed in one place, at the top of the sketch. In the previous sketch, changing the pin number would mean that each place that the pin is used in the sketch would need to be found and modified – three places in the previous sketch. In bigger sketches with many lines of code, this could be difficult and prone to error. As can be seen in the sketch that follows, #define is used to give a meaningful name to a digital pin. The text MY_LED can be any name that is more meaningful than using the pin number. By convention, names defined this way are always written using uppercase letters and underscore characters are used to separate words.

```
#define MY_LED  12

void setup() {
  pinMode(MY_LED, OUTPUT);    // configure digital pin as an output pin
}

void loop() {
  digitalWrite(MY_LED, HIGH); // set pin voltage to 5V on an Arduino Uno
  delay(500);
  digitalWrite(MY_LED, LOW);  // set pin voltage to 0V or GND
  delay(500);
}
```

An alternative to using #define is to use the const keyword which allows the type of data to be specified. Some of the example sketches in the Arduino IDE use this method of defining pins, but don't use the uppercase naming convention. For example, the code below can be used in place of the #define statement in the above sketch.

```
const int MY_LED = 12;
```

But examples in the Arduino IDE may use camel case instead of upper case, such as the following code.

```
const int myLed = 12;
```

2.7.4.2 Pin Current Rating

There is a limit to the amount of current that can be drawn by a load connected to a digital pin configured as an output. This limit is determined by the electrical characteristics of the ATmega328P microcontroller on an Arduino Uno, and is given in the datasheet from the microcontroller manufacturer. Online documentation for Arduino boards is notorious for

promoting bad engineering practice by quoting the absolute maximum rating of components as the normal or usable maximum ratings. On the Arduino Uno page on the Arduino website, DC current per I/O pin (digital pin) was quoted as being 40mA. At some stage this was changed to 20mA which is better, and is the value given on the website at the time of writing this book.

Quoting the maximum current per pin is actually more complex than just quoting a value. The reason for this is that there is an absolute maximum total current of 200mA that can be drawn by the ATmega328P microcontroller. If 20mA was drawn from each of the digital pins, a total of 14 × 20mA = 280mA plus the current that the microcontroller core is using which is way over the 200mA absolute maximum rating. Further to this, certain groups of pins have their own maximum current ratings. What this means is that some pins can be used to draw 20mA from, but not all of them and not too many in a group. This is explained further in the next chapter in section 3.3.1.1, under the Current Limitation Per Pin heading and the sections that follow it. Note also that good design practice dictates that current drawn must not come too close to an absolute maximum rating, but within a certain percentage of it, say 60% to 80% for better reliability – this is usually called overrating.

When a load that draws more current than an output pin can deliver needs to be interfaced to an Arduino Uno, a transistor can be used to interface the heavier load to the Arduino. For even bigger electrical loads, a transistor can be used to switch a relay. Refer to section 3.3.1.1, under the heading “Switching Heavier Loads with Transistors and Relays”, in the next chapter.

2.7.4.3 Input Pins

Digital pins can be configured as input pins in a sketch. An input pin can be used to test whether the voltage applied to it is high (5V) or low (at 0V or GND). This can be used, for example, to see whether a switch that is attached to a pin is open or closed.

By default, when an Arduino is powered up, its pins are configured as input pins. Most sketches that use pins as inputs will explicitly configure the pins as inputs, even though this is not entirely necessary. The ATmega328P microcontroller has internal pull-up resistors attached to each pin that can individually be enabled or disabled. These pull-up resistors are relatively weak, meaning that they have high resistance values and are rated at between 20kΩ and 50kΩ, so have a wide tolerance range. The internal pull-up resistor can be enabled on a pin by passing INPUT_PULLUP to the pinMode() function.

Alternatively pinMode() can be passed INPUT to enable a pin as an input without a pull-up resistor. In this case an external pull-up or pull-down resistor may be needed, depending on what is being interfaced to the pin. See section 3.3.1.2 in the next chapter for examples of interfacing to an input pin. For an example sketch that uses an input pin to test the state of an attached button or switch, select File ► Examples ► 02.Digital ► Button from the top menu of the Arduino IDE.

2.7.4.4 PWM Pins

Some of the digital pins of an Arduino Uno have PWM (Pulse Width Modulation) capabilities and are marked with the tilde symbol (~) on the board. These capabilities are available on certain Arduino pins because the ATmega328P microcontroller has built-in PWM hardware on these pins.

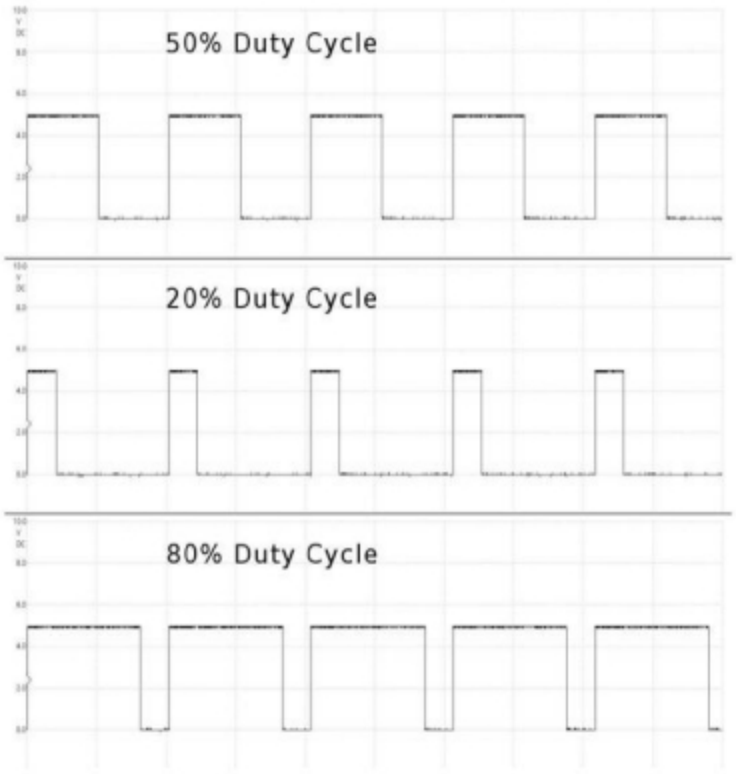


Figure 2.7: PWM Waveforms from an Arduino Uno PWM Pin

PWM allows the brightness of an LED, or the speed of a motor to be controlled. PWM is just a square wave output on a pin that can have its duty cycle changed. Duty cycle is the ratio of on time to off time of the square wave, usually expressed as a percentage.

Figure 2.7 shows an example of the same square wave with its duty cycle adjusted to 50%, 20% and 80%. In each case the frequency of the square wave is exactly the same. A square wave with a 50% duty cycle is created by switching a pin to 5V for a period of time and to GND for the same period of time. Although this is an alternating current waveform, it has an average direct current value. Pulse width modulation works by altering the time that the wave is at 5V to the time that the wave is at GND. If the time that 5V is present is for 80% of the time and GND for only 20% of the time, a higher average voltage will be present on the pin and an attached LED will burn brighter – in this case the duty cycle is said to be 80% because the voltage is at 5V, on or high for 80% of the time. When compared to the same wave at the same frequency, but with the wave at 5V for only 20% of the time and at GND for 80% of the time, a lower average voltage will be present on the pin, and an attached LED will burn more dimly – the duty cycle is said to be 20% for this example because the square wave is on for 20% of the time and off, or at GND, for the remainder of the time.

PWM Example Sketch

In an Arduino sketch, the analogWrite() function is used to control the duty cycle of PWM pins on an Arduino. An example sketch that sets the PWM duty cycle of pin 3 of an Arduino Uno to approximately 50% is shown in the code listing that follows.

```
void setup() {
  pinMode(3, OUTPUT);

  analogWrite(3, 127);
}

void loop() {
}
```

Notice that PWM is set up on the pin in the setup() part of the sketch when analogWrite() is called. The sketch then starts executing the main loop(), which does nothing. At this stage, the PWM waveform set up on the PWM pin will still be present because the internal hardware of the ATmega328P microcontroller is set up to continually output the PWM waveform.

Strictly speaking, pinMode() does not need to be called to set the pin as an output before calling analogWrite(). The first parameter passed to analogWrite() is the digital pin number to produce the PWM waveform on. In the example sketch it is set to 3, which means that the PWM waveform will appear on digital pin 3. At the time of writing, the Arduino IDE does not issue a warning or error message if a digital pin that does not have PWM capabilities is passed to analogWrite(). In this case, a sketch will build and load, but no PWM waveform will be produced on a non-PWM pin.

Calculating PWM Duty Cycle

The second parameter passed to analogWrite() sets its duty cycle, and can be a value between 0 and 255. When set to 0, the output of the PWM pin will be off or at GND level. When set to 255, the PWM pin will be on, or set to 5V. To calculate the value to pass to analogWrite() to set a certain duty cycle as a percentage, divide 255 by 100, and then multiply the result by the desired percentage. For example, to work out the value for a 20% duty cycle, divide 255 by 100 and then multiply by 20: 255 ÷ 100 × 20 = 51. Pass 51 as the second parameter to analogWrite() to set the duty cycle of a PWM pin to 20%, as shown in the code below for pin 3.

```
analogWrite(3, 51);
```

If the result of a duty cycle calculation has a fractional part, then round off to the nearest integer (whole number) value. For example, a 50% duty cycle is calculated as follows: 255 ÷ 100 × 50 = 127.5 which can be rounded to either 127 which would make the duty cycle slightly less than a round 50%, or it can be rounded up to 128 which would make the duty cycle slightly above 50%.

PWM Frequency

On an Arduino Uno, all PWM pins produce a square wave at a frequency of 490Hz, except for pin 5 and pin 6 which operate at a frequency of 976Hz.

PWM LED Control Example

One of the example sketches found in the Arduino IDE, called Fade, demonstrates how to use PWM to continuously change the brightness of an LED. The LED changes from dim to bright and back to dim continually by continually changing the duty cycle of the PWM wave on pin 9 of an Arduino. Find this sketch under File ► Examples ► 0.1Basics ► Fade using the top menu in the Arduino IDE.

2.7.4.5 Serial Port / UART Pins

Arduino Uno pin 0 (RX or receive) and pin 1 (TX or transmit) are connected to the serial port or UART of the ATmega328P microcontroller and are also routed to the ATmega16U2 USB to serial bridge chip. Data that is sent and received over the serial port appears on these two pins. This includes data that is sent to the ATmega328P when a new sketch is being loaded to the Arduino from the Arduino IDE. A 1k resistor between each of these pins and the ATmega16U2 USB to serial bridge chip protect both microcontrollers in case these pins are set up as outputs and data is transmitted from the USB bridge chip, which can cause a short circuit that may otherwise damage these devices. Bear in mind that using these pins as normal digital input or output pins could result in interference with the circuit attached, or voltage levels read in an Arduino sketch, because of the attached ATmega16U2 when it is transmitting. When using these pins as digital pins, the serial port on the Arduino can not be used to send and receive data over the USB link from a sketch. Figure 2.8 shows how these pins are connected on an Arduino Uno.

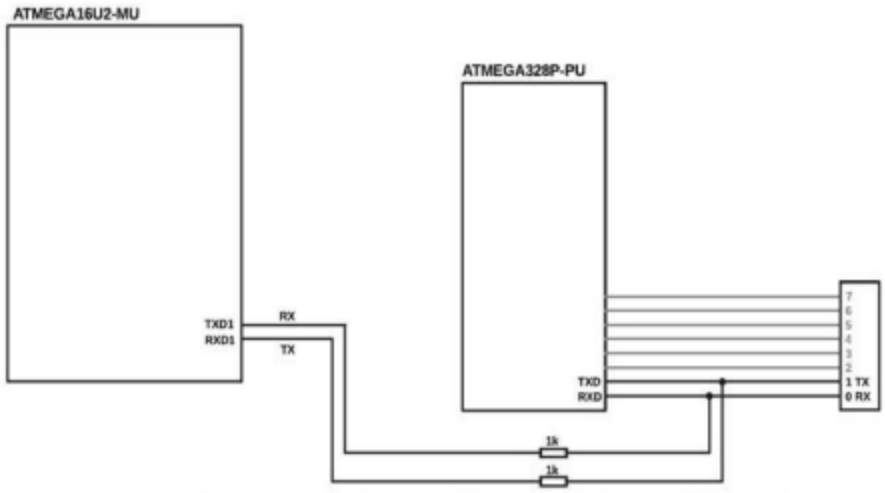


Figure 2.8: Arduino Uno Pin 0 and Pin 1 Connections on the Board

As can be seen in the figure, pin 0 or RX is connected to the ATmega328P RXD pin which is also routed through a 1k resistor to the TXD1 pin of the ATmega16U2. This is so that anything transmitted from the ATmega16U2 TXD1 pin will be received on the ATmega328P RXD pin. Similarly, pin 1 or TX of the Arduino Uno connects to the TXD pin of the ATmega328P which is routed through a 1k resistor to RXD1 of the ATmega16U2. Anything transmitted from the ATmega328P TXD pin will be received on the ATmega16U2 RXD1 pin.

2.7.5 Analog In Pins

Arduino Uno boards have six analog input pins labeled A0 to A5. As with the digital pins, numbering starts at 0 and not 1, which means that A0 to A5 are 6 pins and not 5. Analog pins can read an analog voltage applied to them that can be between 0V to 5V by default.

In the ATmega328P microcontroller the analog pins connect to an ADC (Analog to Digital Converter) that uses a 5V reference voltage to determine the voltage level on the analog pins. It is a 10-bit ADC which means that it will calculate a value between 0 to 1023 that is a representation of the voltage level on one of the analog input pins. Knowing that the reference voltage is 5V and that the resolution of the ADC is 10 bits allows the voltage applied to an analog input pin to be calculated, as explained in section 2.7.5.3 – Calculating Analog In Voltage, that follows.

2.7.5.1 Analog In Example Sketch

A good example sketch that demonstrates how to read an analog value from analog input pin A0 is found under File ► Examples ► 01.Basics ► AnalogReadSerial in the Arduino IDE. The corresponding tutorial page for the AnalogReadSerial example sketch is found on the Arduino website at www.arduino.cc/en/Tutorial/BuiltInExamples/AnalogReadSerial, and shows how to connect a potentiometer to A0 to vary the voltage on this pin. With the potentiometer connected and the AnalogReadSerial sketch loaded, open the Serial Monitor window from the Arduino IDE to see the values that the Arduino is reading from the A0 pin and sending across the serial to USB link. The values will be between 0 and 1023. The serial monitor window is covered in section 1.5.2.3 – Serial Port Demonstration, in Chapter 1.

As can be seen in the AnalogReadSerial sketch, analogRead() is called to get the ADC value from one of the analog inputs. Pass A0 to A5 to this function to select the analog pin to read from.

2.7.5.2 Floating Analog Input Pin

If nothing is attached to an analog pin, in other words no stable voltage level is applied to the pin, it will be 'floating'. When the ADC value is read from a floating pin, the value read will be random and depend on what noise the pin is picking up. If a floating pin is continually read and the value displayed, the value usually keeps changing and appears to randomly jump around.

2.7.5.3 Calculating Analog In Voltage

The analog voltage on an analog pin is calculated by dividing the reference voltage by 1024 (for a 10-bit ADC) and then multiplying the result by the value read on the analog input pin. On the Arduino Uno we know that the default reference voltage is 5V, so the calculation becomes *5 ÷ 1024 × <analog value read on pin> = voltage*. The sketch that follows is a modified version of the AnalogReadSerial example sketch that reads the analog value on pin A0, but this time converts the value to voltage.

```
void setup() {
  Serial.begin(9600);
}

void loop() {
  int sensorValue = analogRead(A0);                    // read ADC value
  float voltage = (5.0 / 1024.0) * (float)sensorValue;  // calculate voltage
  Serial.print(voltage);                               // display voltage
  Serial.println(" V");
  delay(1);
}
```

In the code listing above, the analog input A0 is read as an integer value, which can be any value between 0 and 1023. The voltage is then calculated using floating point mathematics so that the decimal voltage value will be displayed. If integer mathematics were used, the voltage would be displayed as whole numbers only, 0, 1, 2, 3, 4 and 5 with no decimal point.

2.7.5.4 Analog In Pins Used for TWI

Two of the analog input pins are multiplexed with the TWI serial bus of the ATmega328P. This means that these pins can be used either as analog inputs or as TWI pins. TWI or I²C compatible devices can be interfaced to pin A4 (TWI SDA pin) and A5 (TWI SCL pin). TWI and I²C devices are supported by the Wire library in the Arduino IDE. For more information on TWI, see section 3.3.4 in next chapter.

2.7.5.5 Analog In Pins Used as Digital I/O

Analog pins on the Arduino Uno are connected to pins of the ATmega328P that are multiplexed pins. This means that they can be configured as digital input pins, or digital output pins, or can be configured as analog pins attached to the internal ADC. To use analog pins as digital output pins, use the same pin functions as used for digital pins, but reference the pins by their analog pin names. To set up analog pin A0 as a digital output,

call pinMode() and pass it A0 as the pin to configure as an output, as shown in the following line of code.

```
pinMode(A0, OUTPUT);
```

To set the voltage level on the pin to either high or low, call digitalWrite(), using A0 as the target pin again, as shown in the following line of code.

```
digitalWrite(A0, HIGH);
```

In this way the Blink example sketch can be rewritten to flash an LED connected to the A0 pin with a series resistor, as shown in the code listing that follows.

```
void setup() {
  pinMode(A0, OUTPUT);
}

void loop() {
  digitalWrite(A0, HIGH);
  delay(500);
  digitalWrite(A0, LOW);
  delay(500);
}
```

As can be seen in the sketch, the analog pin is simply referenced by its pin name and is used as a digital pin that is driven high and low. To use the pin as a digital input, use pinMode() to set the pin up as an input pin in the same way, again referencing it by its analog pin name.

2.7.6 AREF Pin

An external reference voltage can be applied to the AREF pin that will be used by the ADC when doing analog to digital conversions. This can be used in place of the default 5V analog reference, but must be enabled by calling the analogReference() function which must have EXTERNAL passed to it as shown in the following line of code.

```
analogReference(EXTERNAL);
```

The AREF pin can have a reference voltage in the range of 1V to 5V applied to it. **If using an external reference voltage on AREF, the above line of code must be called before reading an analog value by calling analogRead(). If analogRead() is called first, the**

default reference voltage will be shorted to the voltage that is applied to the AREF pin, possibly damaging the main microcontroller on the Arduino.

There is an option of using a 1.1V internal reference for the ADC by passing INTERNAL to analogReference(). Passing DEFAULT to this function uses the 5V reference which is the default value at power up. For more information on the analogReference() function, see www.arduino.cc/reference/en/language/functions/analog-io/analogreference/
For an example of connecting an external reference voltage to the AREF pin, see section 3.3.10 in the next chapter.

2.8 Programming Headers

Two 6 pin (3 × 2) ICSP programming headers are found on the Arduino Uno. One connects to the main microcontroller and the other to the ATmega16U2 USB bridge microcontroller.

Figure 2.9: Atmel-ICE Connected to the ICSP Header of an Arduino Uno

2.8.1 Programming the Arduino Uno with External Programmer on ICSP

To program a sketch to the Arduino Uno (main microcontroller) using an external USB programmer, such as an Atmel-ICE shown in Figure 2.9, from the Arduino IDE, connect the programmer to the ICSP header at the end of the board, on the opposite end to the USB connector, as can be seen in the figure. In the Arduino IDE, select Tools ► Programmer and then select the correct programmer from the list. A list of all the supported programmers can be seen on this menu. To load a sketch to the Arduino Uno using the programmer instead of the USB connection, select Sketch ► Upload Using Programmer from the top menu in the Arduino IDE. This will erase the bootloader from memory making the full 32K of memory available to the sketch. Note that if Atmel Studio or Microchip Studio is installed on the computer, as is described in Chapter 5, this method of programming will not work from the Arduino IDE because of a driver clash.

Microchip Studio (formerly Atmel Studio) is an example of a full featured C and C++ IDE, that can be used for advanced programming of an Arduino Uno using an external programmer connected to the ICSP header. This is an alternative to using the Arduino IDE.

2.8.2 Restore the Bootloader with IDE and External Programmer

If the bootloader is erased by programming the Arduino Uno using an external programmer, as is described in the previous section, or if a new blank microcontroller is installed in the Arduino Uno, the bootloader can be restored using an external programmer and the Arduino IDE. Connect the programmer to the ICSP header, as is described in the previous section, and as is shown in Figure 2.9. In the Arduino IDE, select the correct programmer under Tools ► Programmer from the top menu. Make sure that the correct Arduino is selected under Tools ► Board in the Arduino IDE. Select Tools ► Burn Bootloader from the top menu to load the bootloader to the main microcontroller. This also sets the fuses in the main microcontroller to the correct values. Again, this will not work if Microchip Studio is installed on the computer because of a driver clash. This is not a problem, because Microchip Studio can be used to restore the bootloader. See Chapter 5 for more information on programming the bootloader using Microchip Studio.

2.8.3 ATmega16U2 ICSP Header

The second ICSP header, found near the USB connector, can be used to program the ATmega16U2 microcontroller. It can be used to update the firmware on this chip, however there is a method called DFU that allows the firmware to be upgraded using the USB connection and some additional software. More information on DFU programming can be found at www.arduino.cc/en/Hacking/DFUProgramming8U2 on the Arduino website and in section 5.1 of Chapter 5 of this book.

2.8.4 ICSP Programming Resources

The following links provide more information on the Atmel-ICE,C programming of Arduino boards using Microchip/Atmel Studio, and Microchip Studio. Also see Chapter 5.

Atmel-ICE – www.microchip.com/developmenttools/ProductDetails/atatmel-ice
C Programming – wspublishing.net/avr-c/
Microchip Studio – www.microchip.com/en-us/development-tools-tools-and-software/microchip-studio-for-avr-and-sam-devices

2.8.5 Using an Arduino as an In-System Programmer

One of the external programmers that can be found under Tools ► Programmer in the Arduino IDE is "Arduino as ISP". This option means that an Arduino can be used to program the bootloader to a second Arduino. Find more information on the Arduino website at www.arduino.cc/en/Tutorial/BuiltInExamples/ArduinoISP

2.9 Shared Pins

Several pins are shared between connectors and functions on Arduino Uno boards. Some of these have already been covered in this chapter, but are included here again to make this section complete and to provide additional details.

2.9.1 Serial Port Pins

Digital pins 0 and 1 are shared with the serial port or UART. RX, or receive, is found on digital pin 0. TX, or transmit, is found on digital pin 1. For more information on the serial / UART pins, see section 3.3.6.

2.9.2 L LED Pin

The on-board L LED is connected to digital pin 13. On revision 3 Arduino Uno boards, this LED is buffered through a unity gain op-amp, which means that it will not interfere with any circuit connected to pin 13. On Arduino Uno boards before R3, a 1k resistor in series with the L LED was connected directly between pin 13 and GND. On these boards, the resistor and LED load may interfere with connected circuits. See section 6.1.2 for the circuit diagram of the unity gain op-amp and L LED.

In addition to being the pin that controls the L LED, pin 13 is the SCK (Serial Clock) pin for the SPI bus – see section 2.9.4 that follows.

2.9.3 TWI or I²C Pins

TWI is a serial bus protocol that uses a clock and data line called SCL (Serial Clock) and SDA (Serial Data) and is compatible with the I²C protocol. On an Arduino Uno, these pins are found on analog input pins A5 and A4. Pin A5 can be used as an analog input, or as SCL for the TWI bus. Pin A4 can be used as an analog input, or as SDA for the TWI bus. When using the Wire library to access TWI devices, the default serial clock frequency is 100kHz which can be changed by calling Wire.setClock().

Arduino Uno revision 3 boards route A5 and A4 to the opposite side of the board near the USB connector. Here the same pins are labeled on the header socket using their TWI names, SCL and SDA. The names are silkscreened on the header on newer R3 boards only. Physically and electrically the SCL pin on one side of the board connects to the A5 pin on the other side of the board. SDA on one side of the board connects to A4 on the other side of the board.

2.9.4 ICSP SPI Pins and Reset

The ICSP header at the end of the board that connects to the main microcontroller shares its pins with digital pins on the header sockets at the edges of the board. These are the pins used for the SPI bus protocol as well as the reset pin. The six pins of the ICSP header are connected as shown in Table 2.4.

Table 2.4: ICSP Pin Functions and Connections

ICSP Pin	Pin Function	Connects To
1	MISO – Master In Slave Out for SPI	Digital Pin 12
2	5V	5V Supply
3	SCK – Serial Clock for SPI	Digital Pin 13
4	MOSI – Master Out Slave In for SPI	Digital Pin 11
5	RESET	Reset Pin and Button
6	GND	GND

In addition to MISO, MOSI and SCK, SPI has a slave select pin (SS) which is available on digital pin 10. Figure 2.10 shows a schematic of the pin numbering of the ICSP header and Figure 2.11 shows the same header on the Arduino Uno board.

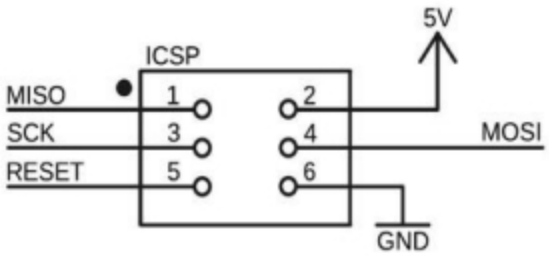


Figure 2.10: ICSP Header Pin Numbers

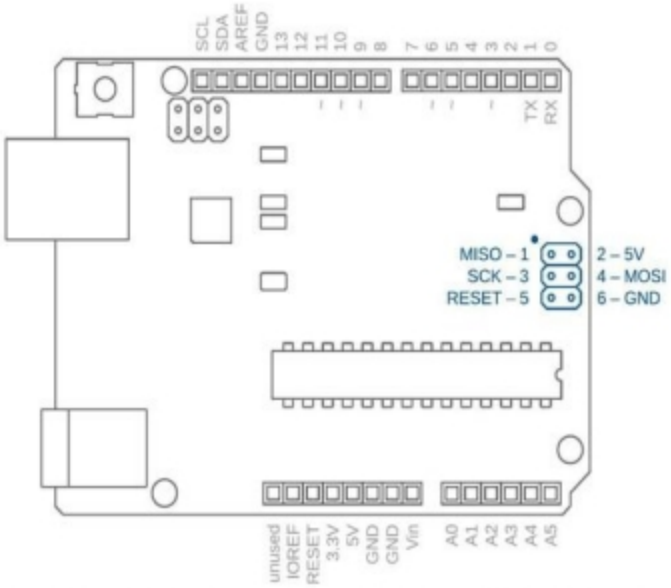


Figure 2.11: ICSP Header Pin Numbers on the Arduino Uno Board

Chapter 3 • Pin Reference and Interfacing

The reference part of this book starts with this chapter, which is an Arduino Uno pin reference with interfacing examples.

Use this chapter as a quick reference to find pin names, alternate pin functions and the mapping of the ATmega328P pins to the Arduino socketed headers and pin header.

Although this book is a hardware manual for the Arduino Uno and not a book that teaches basic electronics, it does include basic interfacing examples to help users with connecting various electronic components to an Arduino. The interfacing examples are used to illustrate various interfacing techniques and principles.

In this Chapter

- Arduino Uno pin names, functions and alternate functions
- Arduino Uno to main microcontroller (ATmega328P) mapping
- Why an LED must be interfaced using a series resistor
- Current sourcing and sinking configurations
- Pull-up and pull-down resistors on inputs
- TWI and SPI serial buses
- Serial port / UART
- Various interfacing examples

3.1 Pin Default and Alternate Functions

Figure 3.1 on the next page shows the default Arduino Uno pin functions as well as the alternate functions such as pins used for TWI and SPI serial buses. This figure is very revealing in that it shows what is really happening with the pins, especially the TWI and SPI pins, where the same pins appear on more than one connector.

3.1.1 Shared TWI Pins

Notice in Figure 3.1 that the TWI pins, SCL and SDA, are shared with analog input pins A5 and A4, but are labeled A5 and A4 on the analog input group of pins. These same pins are routed to the other side of the board where they are labeled SCL and SDA (on Arduino Uno R3 boards, not present on earlier revision boards). In other words there are no additional SCL and SDA TWI pins on Arduino Uno R3 boards, these pins are electrically connected to A5 and A4 on the other side of the board.

Because the TWI pins are shared on an Arduino Uno, this means that if these pins are needed for their TWI function, then only four analog input pins (A0 to A3) are available to use as analog inputs. Bear this in mind if ever designing a shield or building a prototype shield. For compatibility with other shields, avoid the use of A4 and A5 as analog inputs if possible, because other shields may use them as TWI pins. If they do, then they won't be able to be stacked with shields that use A4 and A5 as analog input pins.

3.1.2 Shared SPI Pins

SPI bus pins appear on digital input / output pins 10 to 13. If these pins are used for SPI, then they can no longer be used as input / output pins, and PWM on pins 10 and 11 will no longer be available.

Notice in Figure 3.1 that three of these pins appear on the ICSP header, namely pins 11, 12 and 13. When an external programmer is connected to the ICSP header, it will also be electrically connected to whatever circuit is connected to pins 11, 12 and 13.

As with the TWI pins, when designing a shield or building a prototype shield, avoid using pins 10 to 13 so that the shield will be compatible with shields that use these pins for their SPI functionality. Start designing a shield by using pins that are not multiplexed with other pin functions first. If there are not enough pins, then the SPI or TWI pins can be used, but compatibility with other shields may be lost.

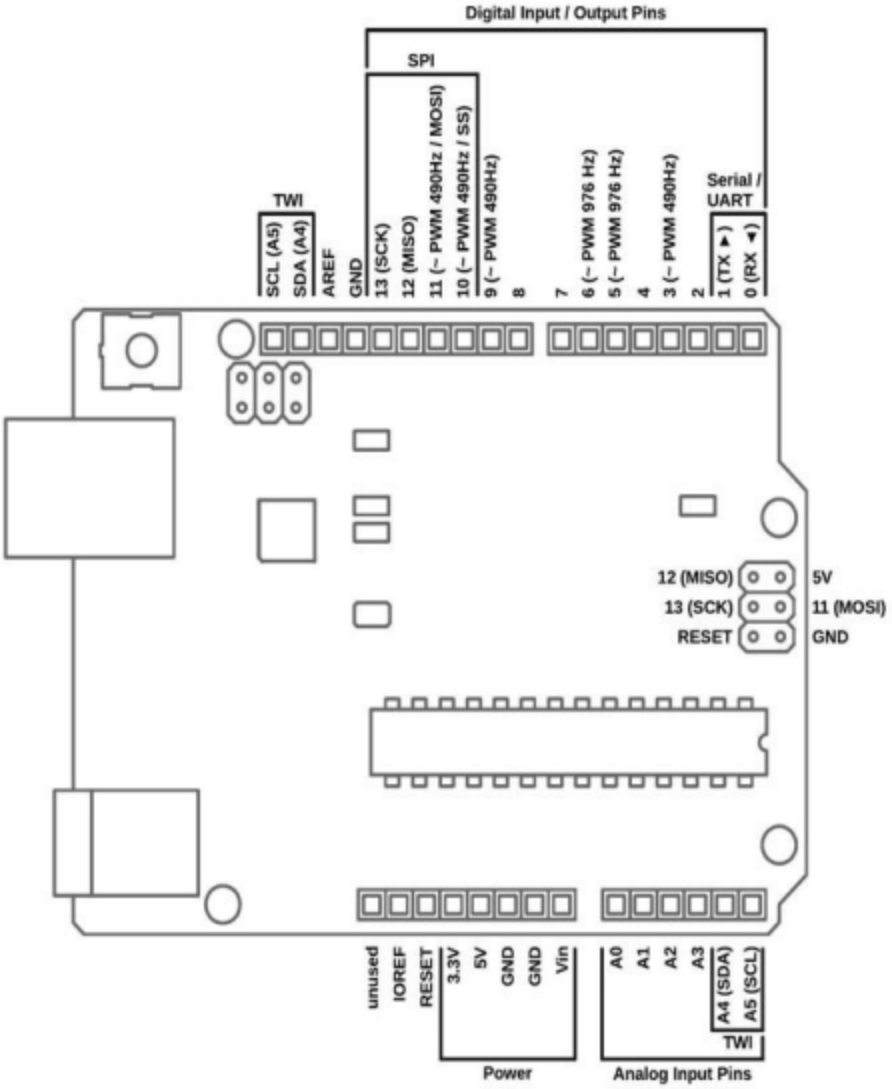


Figure 3.1: Arduino Uno R3 Default Pins and Alternate Pin Functions

3.2 ATmega328P to Arduino Uno Pin Mapping

Figure 3.2 shows the ATmega328P microcontroller chip (not to scale or proportion) with its pin functions and alternate pin functions on the outside labels. The Arduino Uno pins that these microcontroller pins are connected to are shown inside the body of the chip in the figure. Pins 0 to 13 are the digital pins, and A0 to A5 the analog inputs. RESET, AREF, 5V and GND are as labeled on the Arduino Uno board. Only XTAL1 and XTAL2 are not connected to external Arduino pins, but are connected to the on-board ceramic resonator.

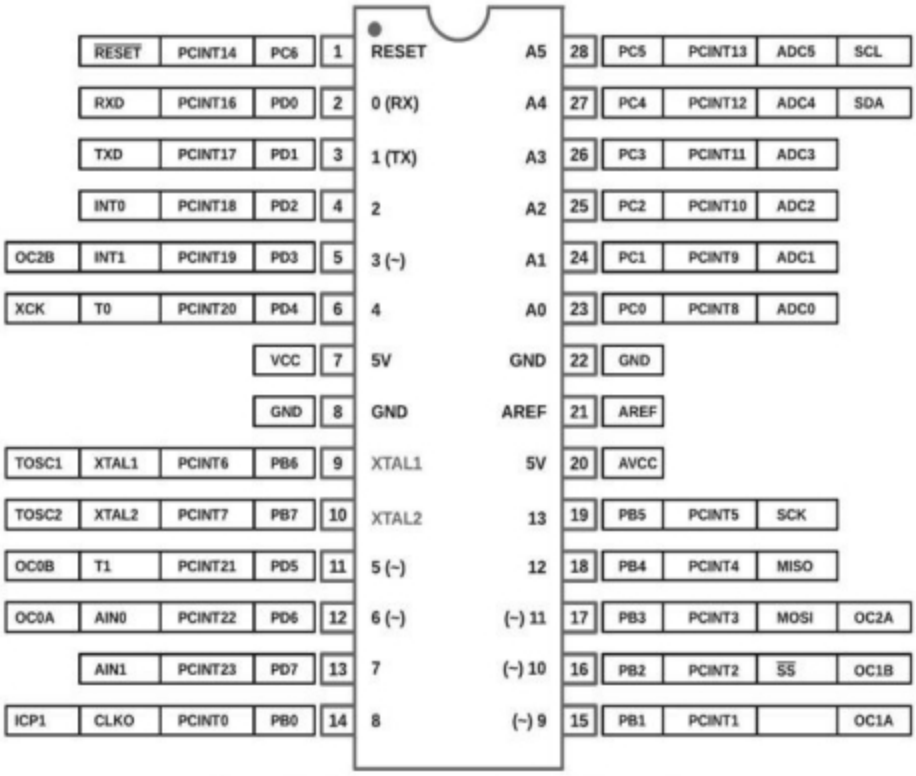

RESET PCINT14 PC6 1 RESET A5 28 PC5 PCINT13 ADC5 SCL
RXD PCINT16 PD0 2 0 (RX) A4 27 PC4 PCINT12 ADC4 SDA
TXD PCINT17 PD1 3 1 (TX) A3 26 PC3 PCINT11 ADC3
INT0 PCINT18 PD2 4 2 A2 25 PC2 PCINT10 ADC2
OC2B INT1 PCINT19 PD3 5 3 (~) A1 24 PC1 PCINT9 ADC1
XCK T0 PCINT20 PD4 6 4 A0 23 PC0 PCINT8 ADC0
VCC 7 5V GND 22 GND
GND 8 GND AREF 21 AREF
TOSC1 XTAL1 PCINT6 PB6 9 XTAL1 5V 20 AVCC
TOSC2 XTAL2 PCINT7 PB7 10 XTAL2 13 19 PB5 PCINT5 SCK
OC0B T1 PCINT21 PD5 11 5 (~) 12 18 PB4 PCINT4 MISO
OC0A AIN0 PCINT22 PD6 12 6 (~) (~) 11 17 PB3 PCINT3 MOSI OC2A
AIN1 PCINT23 PD7 13 7 (~) 10 16 PB2 PCINT2 SS OC1B
ICP1 CLKO PCINT0 PB0 14 8 (~) 9 15 PB1 PCINT1 OC1A

Figure 3.2: ATmega328P to Arduino Pin Mapping

ATmega328P to Arduino pin mapping of Figure 3.2 is generally only needed by advanced Arduino users. For those who want to use alternate hardware functions available on certain pins, and those directly programming the hardware in plain C, the figure is useful to find out which functionality is available on which pin. Figure 3.3 on the next page provides the same information, but from the point of view of the Arduino Uno pins.

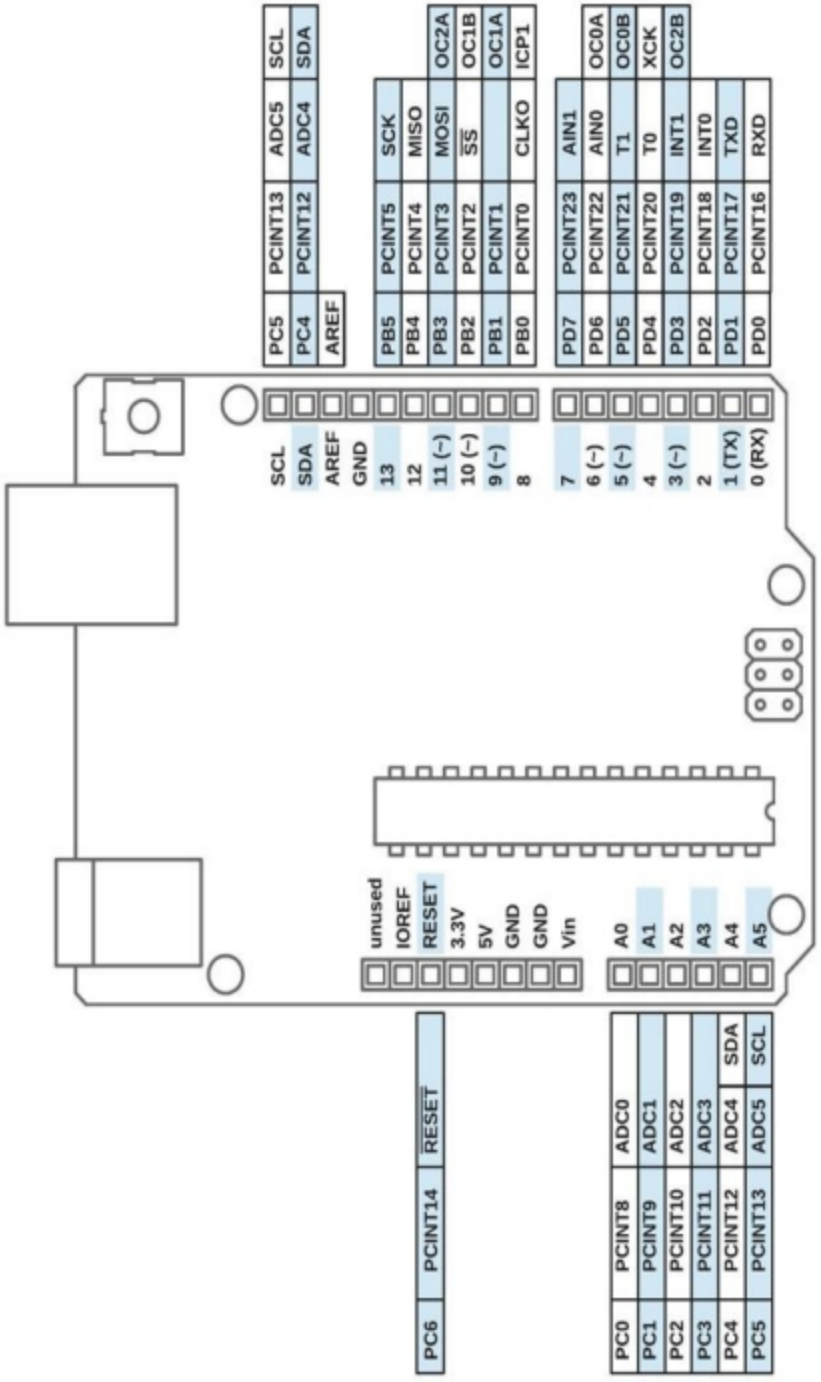


Figure 3.3: Arduino Uno to ATmega328P Pin Mapping

3.2.1 ATmega328P Ports

ATmega328P microcontroller I/O, or input/output pins, are grouped into 8-bit ports that are each labeled with a letter of the alphabet. Three ports are available on the ATmega328P, namely port B (PBx pins), port C (PCx pins) and port D (PDx pins), where x is a number from 0 to 7. These port pin numbers can be seen in the previous two figures, for example PD2, or port D pin 2, maps to Arduino Uno digital I/O pin 2. When programming in plain C, ports or individual pins on ports can be programmed by knowing their port and pin names. When programming using the Arduino IDE with the Arduino library functions, these same port pins are accessed using their Arduino Uno pin names. The real microcontroller pin names are hidden by software.

Although the ports are 8 bits wide, all bits from a port may not be available on pins of the microcontroller, because there may not be enough pins on the microcontroller. On the 28-pin PDIP version of the ATmega328P, the full port B is available, but on an Arduino Uno, pins PB6 and PB7 connect to an external ceramic resonator used for generating the square wave clock frequency for driving the microcontroller. Port C only has 7 pins available, PC0 to PC6. On an Arduino Uno PC6 is dedicated to the reset function. Finally port D has all 8 pins available on the PDIP package, PD0 to PD7.

3.2.2 ATmega328P Alternate Pin Functions

Each additional label next to each pin in Figure 3.2 and Figure 3.3 denotes an alternate function that is multiplexed on the pin. Multiplexed means that the pin can be configured in software to use one of the alternative pin functions. For example Arduino Uno pin A5 can be used as a general purpose I/O port pin PC5. The same pin can use the pin change interrupt function (PCINT13), or it can be used as an analog input pin (ADC5), or it can be used as TWI serial clock (SCL). Using an alternative pin function essentially connects the pin to a hardware device inside the microcontroller that performs the function, such as the hardware that handles TWI serial communications. This hardware is configurable by software, which is usually handled by Arduino libraries used in the Arduino IDE.

Official Arduino libraries don't support all possible ATmega328P pin functions. These pin functions can still be used by programming them in C or C++, typically by more advanced users or hackers. More information on each pin function can be found in the ATmega328 datasheet from the page www.microchip.com/wwwproducts/en/ATmega328P on the Microchip website. A link to the datasheet can be found on this page.

3.3 Pin Types and Interfacing

In this section pins are grouped by types or functions, such as digital, analog, PWM, TWI, and so on. Basic interfacing circuits and principles are presented in each section where appropriate.

3.3.1 Digital Input / Output Pins

Arduino Uno pins 0 to 13 are digital I/O pins and can individually be configured as input pins or output pins in an Arduino sketch. Pins 0 and 1 are connected to the USB to serial bridge chip via two 1k resistors, so may not be suitable as I/O pins, depending on the application. Although analog input pins A0 to A5 are labeled as ANALOG IN pins on an Arduino Uno, they can be used as digital input and output pins as well.

3.3.1.1 Pins as Outputs

Pins are configured as outputs in the setup() part of an Arduino sketch. Call the pinMode() function to set an Arduino pin up as an output that can be used to switch an attached device on and off. This function has the following format when setting up a pin as an output.

```
pinMode(<pin number>, OUTPUT);
```

Where ***<pin number>*** can be one of the following.

0 to 13 – for digital pins.
A0 to A5 – to set up an analog pin as a digital output.
LED_BUILTIN – set up the default L LED pin as an output.

Examples:

```
pinMode(2, OUTPUT);   // set up digital pin 2 as an output
```

```
pinMode(A0, OUTPUT);  // set up analog input pin A0 as a digital output
```

```
pinMode(LED_BUILTIN, OUTPUT);  // set up pin attached to L LED as an output
```

Why an LED Needs a Series Resistor

LEDs are diodes that have a certain forward voltage rating and maximum forward current. If a voltage higher than the maximum forward voltage is applied directly between the leads or pins of an LED, it will try to draw more than its maximum rated forward current. This can cause the LED to burn out, and/or the device that is driving it, such as the microcontroller on an Arduino. Putting a resistor in series with an LED limits the current that the LED can draw, preventing it from burning out and damaging the microcontroller.

Figure 3.4 shows an LED with a series resistor. If the forward voltage, V_F, across the LED is 2V, then the voltage across the resistor will be the supply voltage minus the LED V_F voltage. In this example it is 5V – 2V = 3V dropped across the resistor, where 5V can be directly supplied from a power supply or switched onto the circuit from an Arduino pin. As can be seen in this figure, excess voltage is dropped across resistor R1, preventing too much voltage being applied directly across the LED.

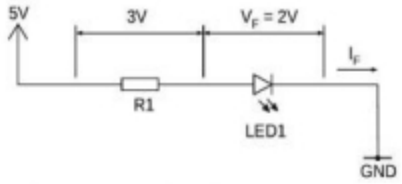


Figure 3.4: LED and Series Current Limiting Resistor

The actual LED forward voltage depends on the type and color of the LED. A red diffused LED typically has a forward voltage of around 2V, while some blue and white LEDs can have forward voltages of 3V or higher. Forward voltage of an LED is also affected by the forward current, I_F, flowing through the LED – forward voltage will typically increase slightly as forward current is increased. Refer to the LED manufacturer's datasheet for the actual electrical characteristics of an LED. As an example, a red diffused 5mm LED typically has the following characteristics.

Forward Voltage:	1.7V to 2V (@ 20mA forward current)
Forward Current:	20mA
Peak Forward Current:	30mA
Maximum Reverse Voltage:	5V

How to Calculate a LED Current Limiting Series Resistor

Before calculating the value of a series current limiting resistor for an LED, forward current must be selected, which will typically be a value in milliamps (mA). The more current that flows through the LED, the brighter it will burn. Limiting factors are the maximum forward current of the LED and the amount of current that the device driving the LED can deliver. If the maximum forward current of an LED is 20mA and the maximum current that an Arduino Uno pin can deliver is also 20mA, then for good design practice, a value less than 20mA must be selected. Forward voltage of the LED must also be known before it is possible to calculate the series resistor value. Two examples follow that show how to calculate the series resistor value. A third and fourth example show how to calculate the current flowing through the LED and series resistor if the value of the resistor is known.

Example 1

It is decided to calculate the series resistor value using a forward current of 10mA, which is well within the 20mA limit, and a forward LED voltage of 2V. To do the calculation, we use Ohm's law, but first calculate the voltage drop across the resistor.

An Arduino Uno operates at 5V, and switching a digital pin high switches 5V onto the pin. The voltage dropped across the resistor will then be 5V minus the 2V voltage drop across the LED:

5V – 2V = 3V

To calculate the resistance of the series resistor, divide the resistor voltage drop by the desired forward current. In this example it is 3V divided by 10mA:

3V ÷ 10mA = 300Ω
OR
3V ÷ 0.01A = 300Ω
(10mA = 0.01A)

A 300Ω series resistor will limit the LED current to 10mA.

Example 2

It is decided that the LED from the previous example is not bright enough for the application, so the LED forward current is changed to 16mA, which is still within the 20mA

limit. The same LED with a forward voltage of 2V is used, so the voltage drop across the series resistor is still 3V. The resistor is calculated as follows:

3V ÷ 16mA = 187.5Ω
OR
3V ÷ 0.016A = 187.5Ω

As 187.5Ω is not a standard resistor value, use 180Ω, 200Ω or 220Ω instead.

Example 3
From the previous example, if a 220Ω resistor is the only value close to 187.5Ω that is available, the current flowing through the 220Ω series resistor and LED can be calculated as follows, if the voltage drop across the resistor is 3V:

I = V ÷ R (Current equals Voltage divided by Resistance)

I = 3V ÷ 220Ω
I = 0.0136A OR 13.6mA

Example 4
In a circuit diagram, an LED and 470Ω resistor are connected in series. Assuming the LED forward voltage is 2V and the supply voltage is 5V, the current through the LED can be calculated as follows:

Voltage across resistor = 5V – 2V = 3V

I = V ÷ R
I = 3V ÷ 470Ω
I = 0.0064A OR 6.4mA

All of the series resistor current calculations are approximate, because resistors have certain tolerances, such as 5% or 1%. This means that the resistance values of these resistors could be higher or lower by their given tolerance. LED voltages are also not exact values, but could be higher or lower than the value used in calculations. Finally power supply values may not be exactly 5V. The main point is that calculated current values must be below the maximum LED and Arduino pin current ratings for safety.

Current Sourcing and Current Sinking

There are two ways that an LED and series resistor, or other load such as a buzzer, can be connected to an Arduino or other electronic system, namely current sourcing and current sinking. How they are wired determines how they are turned on and off in software sketches. In a current sourcing configuration, an Arduino pin is switched high to switch the load on and low to switch the load off. For a current sinking configuration, the opposite is true – the pin is switched high to switch the load off and low to switch the load on. Both configurations are explained in the sections that follow. In the figures that follow, it does not matter if the LED and series resistor are swapped. It does matter that the LED is connected the correct way around, as current can only flow one way through an LED.

Current Sourcing

Figure 3.5 shows an LED and series resistor connected to an Arduino Uno in a current sourcing configuration. When Arduino Uno pin 2 is switched high, the Arduino sources current that flows through the series resistor and LED, switching the LED on. When the pin is switched low, the LED switches off.

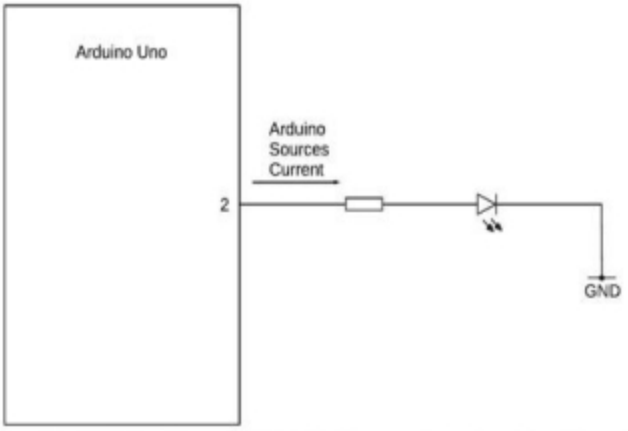


Figure 3.5: Arduino Uno and LED in Current Sourcing Configuration

In a sketch, pin 2 is first set up as an output pin as follows.

```
pinMode(2, OUTPUT);
```

To switch the LED on, the pin is switched high as shown in the following line of code.

```
digitalWrite(2, HIGH);    // switch LED on (current sourcing)
```

To switch the LED off, the pin is switched low as shown below.

```
digitalWrite(2, LOW);    // switch LED off (current sourcing)
```

The built-in L LED on Arduino Uno boards is wired to pin 13 of the Arduino in a current sourcing configuration and should be the most familiar way of switching an LED on and off for most users.

Current Sinking

Figure 3.6 shows an Arduino Uno and LED connected in a current sinking configuration. The anode of the LED is connected to 5V, this means that it can only switch on if the other side of the LED, or cathode is connected to GND through the series resistor. When connected to an Arduino, as shown in the figure, the Arduino pin can be switched to either 5V or GND. Switching the pin to 5V, or high, switches the LED off because both sides of the LED series resistor circuit are at this same potential. The only way to switch the LED on is to switch the Arduino pin low, which provides GND to the other side of the circuit. Current then flows from the 5V supply, through the LED and series resistor to GND that is supplied by the microcontroller on Arduino pin 2.

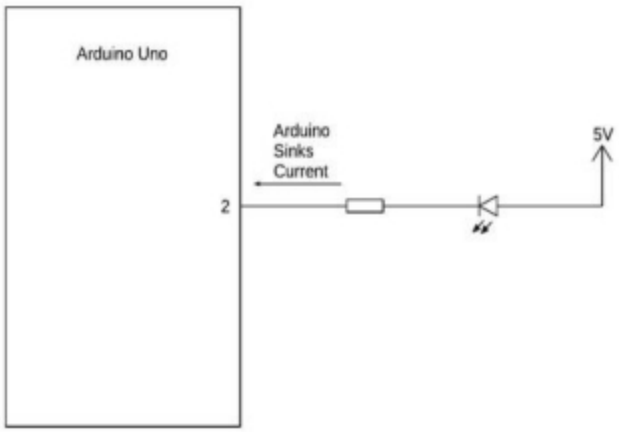


Figure 3.6: Arduino Uno and LED in Current Sinking Configuration

As with current sourcing, current sinking requires the Arduino pin to be set up as an output. In a sketch, pin 2 is first set up as an output pin as follows.

```
pinMode(2, OUTPUT);
```

To switch the LED on, the pin is switched low as shown in the following line of code.

```
digitalWrite(2, LOW);    // switch LED on (current sinking)
```

To switch the LED off, the pin is switched high as shown below.

```
digitalWrite(2, HIGH);   // switch LED off (current sinking)
```

Current sinking may seem strange at first, but is a legitimate way to wire a load to an Arduino or other electronic device. Users are usually accustomed to high being on and low being off, current sinking reverses this logic. Bear this in mind if ever coming across a circuit or shield that has an LED or other device connected in a current sinking configuration.

Current Limitation Per Pin

On the Arduino Uno R3 page on the Arduino website, the technical specifications give the current limit per pin as 20mA. The ATmega328P datasheet gives the absolute maximum current per pin as 40mA. Further to this, the datasheet gives the absolute maximum total current that can be drawn from the device as 200mA. No device should be operated at its absolute maximum values, but rather well below them to avoid destroying the device and for long term reliability. It is put this way in the datasheet: *"Exposure to absolute maximum rating conditions for extended periods may affect device reliability."* In addition to these operating conditions, there is a limit of 100mA that certain groups of pins can source and sink as shown in the following sections.

I/O Port Current Source Limits

I/O port current source limits are described as follows in the ATmega328P datasheet, as they apply to the 5V Arduino Uno. This looks slightly different in the datasheet because the pins and voltage levels that are not applicable to the Arduino Uno have been removed.

Although each I/O port can source more than the test conditions (20mA at VCC = 5V) under steady state conditions (non-transient), the following must be observed:

- The sum of all IOH, for ports C0 – C5, D0 – D4, RESET should not exceed 100mA. (Cx and Dx pin numbers are the same as PCx and PDx)

- The sum of all IOH, for ports B0 – B5, D5 – D7, XTAL1, XTAL2 should not exceed 100mA. (Bx and Dx pin numbers are the same as PBx and PDx)

Figure 3.7 shows the two groups of pins that can source 100mA per group. The wide arrow near the pin labels shows that the current is being sourced from the Arduino – current flows out of the pins. The group at the left of the figure includes XTAL1 and XTAL2 which are dedicated oscillator pins on the Arduino Uno.

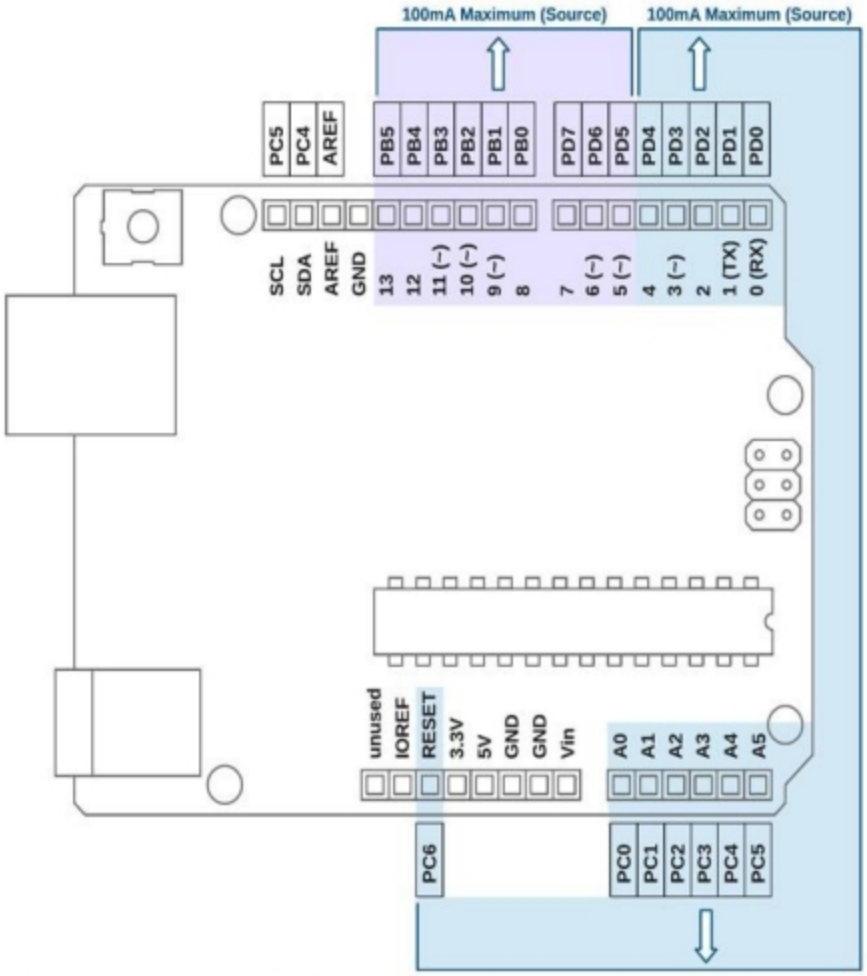


Figure 3.7: Arduino Uno Pin Groups that can Source a Maximum of 100mA Each

The group at the right of Figure 3.7 includes the reset pin and the analog input pins. If these pins are used for their default Arduino Uno pin functions, they will be used as inputs and can be ignored when calculating the maximum current per group. In other words Arduino Uno digital pins 0 to 4 can then source a maximum of 100mA. If analog pins A0 to A5 are used as digital output pins, then they need to be included in the group.

From Figure 3.7, it can be seen that although 20mA can be sourced from each digital pin, this can not be done from all of the pins simultaneously. For example the group of pins on the left of the figure (digital pins 5 to 13) are 9 pins in total. If each pin is set up as an output and 20mA is drawn from each pin, the total current drawn will be 9 × 20mA = 180mA, which is way over the 100mA maximum limit. Good design practice dictates that current must not be drawn too near to the 100mA absolute maximum. Using 80% of absolute maximum is a reasonable choice, which of course would be 80mA per group of pins. In this case an average of 80mA ÷ 9 = 8.9mA could be drawn from each pin simultaneously. Because the absolute maximum that can be drawn per pin is 40mA, two pins in a group could be used to source 30mA each, leaving 20mA available to be drawn from the remainder of the group if working with an 80% or 80mA limit for the group.

With the above understanding of how the pin groups work, use Figure 3.7 to see which pins a circuit can be connected to, depending on how much current each pin in a group is required to source. If, for example, 5 pins are required to source 20mA each, then make sure that they are spread between the two pin groups so as not to draw too much current from each group. After designing a circuit that connects to an Arduino Uno, use the figure to see which pins are being used to source current, and then make sure that the current per pin group does not come too close to, or exceed the 100mA limit.

I/O Port Current Sink Limits

When using output pins on an Arduino Uno to sink current, there are 3 groups of pins that have an absolute maximum current sink limit of 100mA, but this does not mean that the Arduino or microcontroller can sink 300mA of current – there is still a 200mA total limit.

As with current sourcing, current sinking pin groups have been taken from the datasheet and slightly modified to suit the Arduino Uno and are presented below.

Although each I/O port can sink more than the test conditions (20mA at VCC = 5V) under steady state conditions (non-transient), the following must be observed:

- The sum of all IOL, for ports C0 – C5 should not exceed 100mA.
- The sum of all IOL, for ports B0 – B5, D5 – D7, XTAL1, XTAL2 should not exceed 100mA.
- The sum of all IOL, for ports D0 – D4, RESET should not exceed 100mA.

Figure 3.8 shows the 3 pin groups that can sink a total absolute maximum of 100mA each. The wide arrows in the figure represent current sinking in the Arduino Uno – current flows into the Arduino. Current sinking limits work the same way as current sourcing limits. Don't allow more than 100mA of current to be sunk by each group of pins. Rather work within a reasonable percentage of the absolute maximum, say 80%. Don't allow the Arduino Uno to sink more than 200mA total across all groups of pins. Use the figure to make sure that a circuit is not sinking current near the maximum limit for a group of pins.

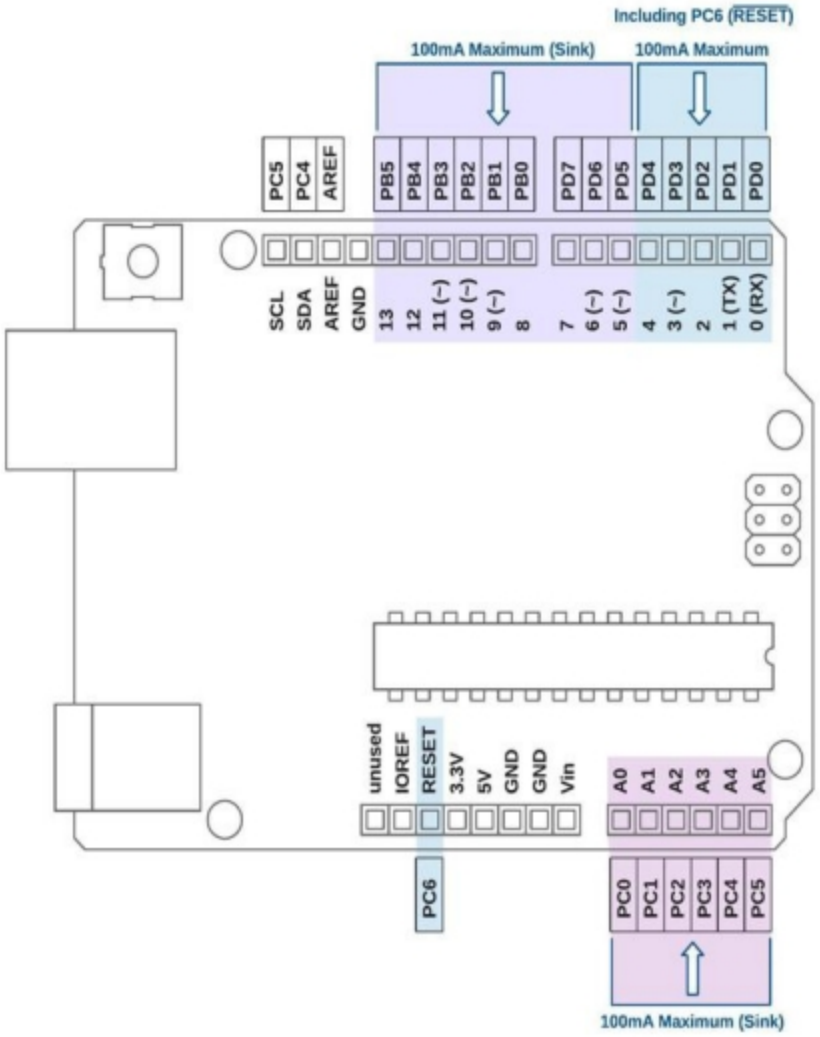


Figure 3.8: Arduino Uno Pin Groups that can Sink a Maximum of 100mA Each

Switching Heavier Loads with Transistors and Relays

When it is required to use an Arduino to switch a heavier load than an Arduino pin can handle, two basic options are to use a transistor, or a transistor and a relay. Other options are available such as optocouplers or opto-isolators, solid state relays, thyristors, and similar devices. This book looks only at basic interfacing and does not cover more advanced devices.

Figure 3.9 shows a basic transistor interface circuit using an NPN BJT (Bipolar Junction Transistor). The transistor is used not only to switch on a device that draws more current than an Arduino Uno pin can deliver, but also to switch on a device that operates at a higher voltage than the Arduino. In this example a buzzer is powered from a 12V external supply and switched on by the transistor that is controlled by the Arduino.

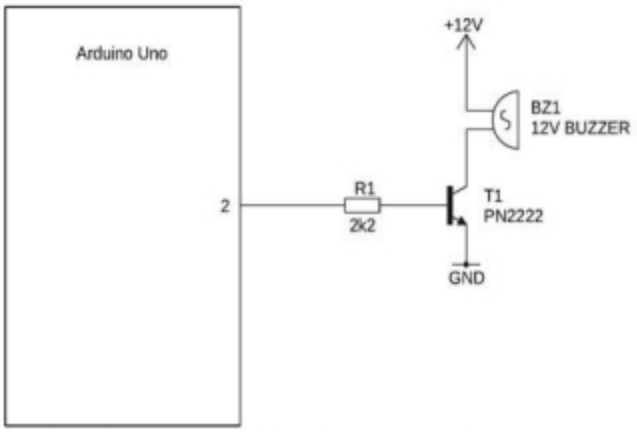


Figure 3.9: Switching on a 12V Buzzer using an Arduino Uno and Transistor

When the Arduino Uno pin is set to a high level in a sketch, current flows from the pin through resistor R1, into the base of the transistor and out the emitter pin of the transistor to GND. This switches the transistor on causing current to flow from the 12V supply through the buzzer, into the collector pin of the transistor, out of the emitter of the transistor and finally to GND. Switching the Arduino pin to a low level stops current flowing into the base of the transistor which switches the transistor, and therefore the buzzer off.

The positive lead of the external 12V power supply must be connected to the positive pin of the buzzer and the negative or GND of the power supply must be connected to the GND

of the Arduino. A transistor such as a PN2222 can easily switch a load of 100mA or more, which enables it to be used with a number of small DC devices.

When an Arduino Uno is required to switch more current than a simple transistor can handle, or if it needs to switch an AC load, then a relay is an option. A relay will usually be driven by a transistor as shown in Figure 3.10.

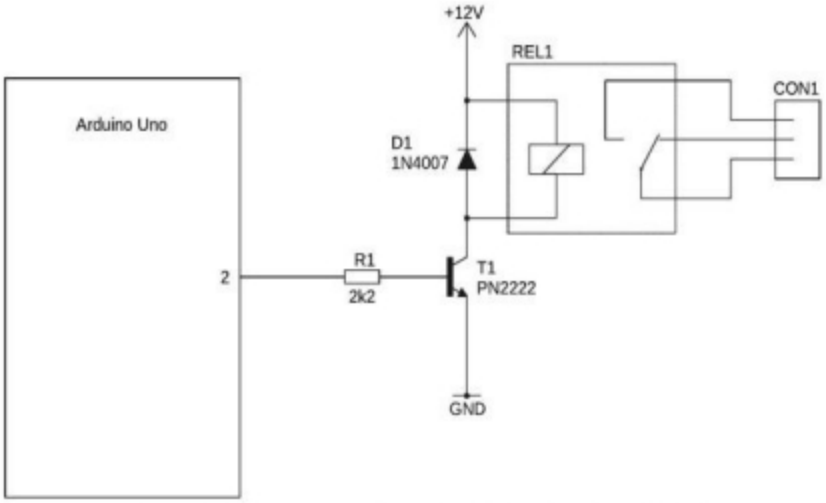


Figure 3.10: Using an Arduino Uno to Switch a Relay On

A relay has a coil, that when activated switches a set of contacts. A simple relay may have one set of normally open contacts that can be used as a switch in an external circuit, for example to switch an AC bulb on. Other relays may have a normally open and normally closed set of contacts, or even more than one set of contacts.

The coil of a relay is basically an electromagnet – a coil of wire wrapped many times around a metal core. When a DC voltage is applied across the coil, the metal core magnetizes enough to close the set of relay contacts. When the voltage is removed from the coil, the magnetic field that exists in the core breaks down, inducing a high voltage back into the coil. For this reason a protection diode is placed between the relay coil pins, as diode D1 in Figure 3.10 shows. This prevents the high voltage produced by the decaying magnetic field in the relay coil from destroying the transistor and Arduino.

As mentioned before, this book is an Arduino Uno reference manual and so does not go into a lot of detail on interfacing, but rather just uses some simple examples to show

interfacing principles. For more details on interfacing Arduino boards and microcontrollers, refer to the appropriate texts or other sources such as the *Interfacing With Hardware* page playground.arduino.cc/Main/InterfacingWithHardware/ in the playground area on the Arduino website.

3.3.1.2 Pins as Inputs

Pins are configured as inputs in the setup() part of an Arduino sketch. Call the pinMode() function to set an Arduino pin up as an input that can be used to read the logical state of a pin (high or low, on or off), or the state of a device attached to a pin. Input pins can be set up to use internal pull-up resistors that are internal to the ATmega328P microcontroller, or they can be set up not to use internal pull-up resistors. Pull-up and pull-down resistors are explained further in this section.

The pinMode() function has the following format when setting up a pin as an input.

```
pinMode(<pin number>, <input type>);
```

Where ***<pin number>*** can be one of the following.

0 to 13 – for digital pins.

A0 to A5 – to set up an analog pin as a digital input.

Input type ***<input type>*** can be one of the following.

INPUT – input without internal pull-up resistor.

INPUT_PULLUP – input with internal pull-up resistor enabled.

Examples:

```
pinMode(2, INPUT);   // set up digital pin 2 as an input (no internal pull-up)
```

```
pinMode(A0, INPUT);  // set up analog input pin A0 as a digital input
```

```
pinMode(7, INPUT_PULLUP);  // digital pin 7 as an input with internal pull-up
```

An Arduino pin that is interfaced to a switch needs either a pull-up or pull-down resistor connected to it. If no resistor is used, the pin will be “floating" which means that the state of the switch (on or off) can not be reliably read by the Arduino.

Pull-down Resistor

A pull-down resistor pulls an Arduino Uno digital pin down to GND, as shown in Figure 3.11. A switch attached to the pin will connect the pin to 5V when it is closed. When the switch is open and the state of the pin is read in a sketch, it will be low because the pull-down resistor is pulling it low, or to GND. When the switch is closed, and the state of the switch is read by the Arduino, the state of the switch will be high. The switch does not short 5V to GND because of the pull-down resistor.

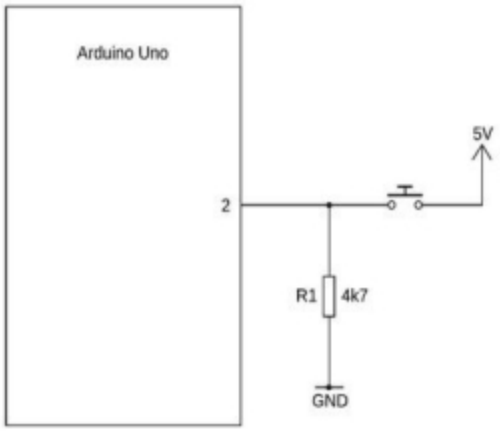


Figure 3.11: Switch Interfaced to an Arduino Uno with a Pull-down Resistor

In the sketch code listing that follows, the state of Arduino Uno digital pin 2 is read, and the on-board L LED is switched on if the switch is closed. If the switch is open, the LED is switched off.

```
void setup() {
  pinMode(LED_BUILTIN, OUTPUT);
  pinMode(2, INPUT);
}

void loop() {
  if (digitalRead(2)) {
    // pin state is high or 1, switch is closed (pull-down resistor)
    digitalWrite(LED_BUILTIN, HIGH);  // switch L LED on
  }
  else {
    // pin state is low or 0, switch is open (pull-down resistor)
    digitalWrite(LED_BUILTIN, LOW);   // switch L LED off
  }
}
```

As can be seen in the code listing, digitalRead() is used to read the state of the switch, or pin that the switch is connected to, and is passed the pin number to read. This function returns 0 if the logic level on the pin is low (switch is open) or 1 if the logic level on the pin is high (switch is closed).

For a similar sketch, open the DigitalReadSerial example sketch found under File ▸ Examples ▸ 01.Basics ▸ DigitalReadSerial from the top menu of the Arduino IDE. Find the tutorial page at www.arduino.cc/en/Tutorial/BuiltInExamples/DigitalReadSerial on the Arduino website. When running this example, open the Serial Monitor window in the Arduino IDE to see the state of the switch.

Pull-up Resistor

An alternative to using a pull-down resistor when interfacing a switch to an Arduino Uno is to use a pull-up resistor as shown in Figure 3.12. When the switch is open and the state of the pin is read in a sketch, it will be high or 1. This is because the resistor is pulling the pin up to 5V or high. When the switch is closed, it connects the pin to GND or low – in a sketch the state of the switch will now be read as low or 0. When the switch is closed, it does not short 5V to GND because of the pull-up resistor.

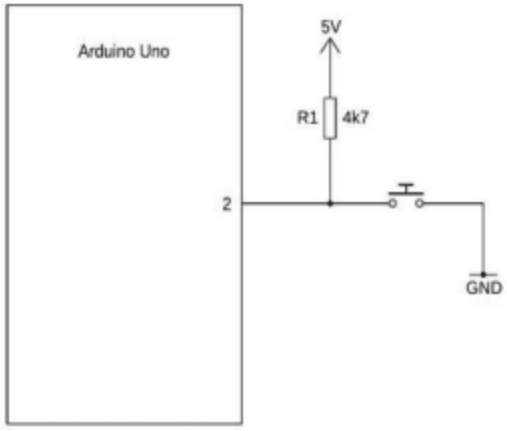


Figure 3.12: Switch Interfaced to an Arduino Uno with a Pull-up Resistor

In the sketch code listing that follows, it can be seen that the logic is reversed when using a pull-up resistor with a switch instead of a pull-down resistor. This means that when a value of 1 or high is read from pin 2, the switch is open and not closed. When a value of 0 or low is read, the switch is closed, because the input pin is pulled down to GND.

```
void setup() {
  pinMode(LED_BUILTIN, OUTPUT);
  pinMode(2, INPUT);
}

void loop() {
  if (digitalRead(2)) {
    // pin state is high or 1, switch is open (pull-up resistor)
    digitalWrite(LED_BUILTIN, LOW);    // switch L LED off
  }
  else {
    // pin state is low or 0, switch is closed (pull-up resistor)
    digitalWrite(LED_BUILTIN, HIGH);   // switch L LED on
  }
}
```

As can be seen in the sketch, digitalRead() returns 1 when the switch is open, so the L LED is switched off. When the switch is closed, digitalRead() returns 0, so the L LED is switched on. When this pull-up configuration is used with the DigitalReadSerial example sketch mentioned in the previous section, the logic is reversed. 1 is displayed in the Serial Monitor window when the switch is open, it then changes to 0 when the switch is closed.

Internal Pull-up Resistors

The internal resistor on a pin can be enabled in a sketch, in which case a switch can be interfaced to the pin by connecting it between the pin and GND as shown in Figure 3.13.

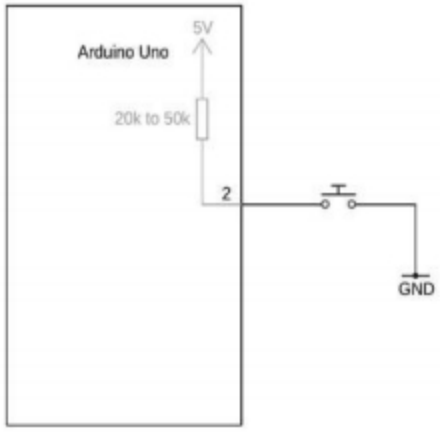


Figure 3.13: Switch Interfaced to an Arduino Uno with Internal Pull-up Resistor Enabled

Internal pull-up resistors are fairly weak which means that they have high resistance values. They also have a very wide tolerance range, and can be between 20kΩ and 50kΩ. A weak pull-up resistor will be more susceptible to electrical noise, so should only be used in environments that are not excessively electrically noisy.

The sketch that follows is exactly the same as the previous sketch, except that pin 2 is set up using INPUT_PULLUP as the second parameter passed to pinMode() which enables the internal pull-up resistor. The logic of the sketch works exactly the same as the previous sketch that was used with an external pull-up resistor.

```
void setup() {
  pinMode(LED_BUILTIN, OUTPUT);
  pinMode(2, INPUT_PULLUP);
}

void loop() {
  if (digitalRead(2)) {
    // pin state is high or 1, switch is open (pull-up resistor)
    digitalWrite(LED_BUILTIN, LOW);    // switch L LED off
  }
  else {
    // pin state is low or 0, switch is closed (pull-up resistor)
    digitalWrite(LED_BUILTIN, HIGH);   // switch L LED on
  }
}
```

3.3.2 PWM Pins

Pulse Width Modulation (PWM) and PWM pins are explained in section 2.7.4.4 of the previous chapter. PWM is only available on the Arduino Uno pins marked with a tilde (~) symbol on the board, and as shown in Figure 3.14 on the next page. The analogWrite() function is used to generate a PWM waveform and change the PWM duty cycle on the selected PWM pin. A good example that demonstrates PWM is the Fade example sketch found under File ► Examples ► 01.Basics ► Fade on the top menu of the Arduino IDE. Interfacing an LED to a PWM pin is the same as interfacing an LED to a digital output pin, it must be connected using a current limiting series resistor, as has already been explained in this chapter. Figure 3.5 and Figure 3.6 show the current sourcing and current sinking configurations – use the same circuits for PWM control of an LED, but make sure that the circuit is connected to one of the PWM pins and not pin 2 as is shown in the figures.

The Arduino reference has the following to say about Arduino Uno pins 5 and 6 when being used as PWM pins: *"The PWM outputs generated on pins 5 and 6 will have higher-than-expected duty cycles. This is because of interactions with the millis() and delay() functions, which share the same internal timer used to generate those PWM outputs. This will be noticed mostly on low duty-cycle settings (e.g. 0 – 10) and may result in a value of 0 not fully turning off the output on pins 5 and 6." (Taken from www.arduino.cc/reference/en/language/functions/analog-io/analogwrite/)*

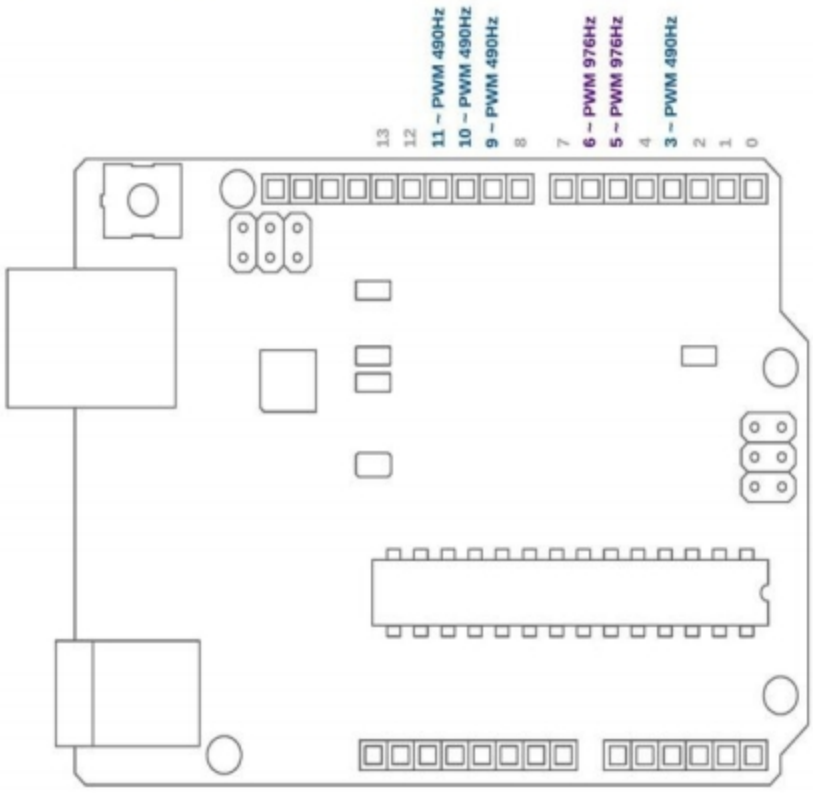


Figure 3.14: Arduino Uno PWM Pins and Frequencies

3.3.3 Analog Pins

Analog input pins A0 to A5 can read analog voltage values between 0V and 5V by default. Whatever device is interfaced to any of these pins must keep the voltage in this range to avoid damaging the Arduino or ATmega328P microcontroller. Not much more can be said in this section than has already been said about analog input pins in section 2.7.5 from the previous chapter. Refer to that section for information on using the analog input pins

and refer to sub-section 2.7.5.1 – Analog In Example Sketch for an example of interfacing a potentiometer to an analog input. Sub-section 2.7.5.3 – Calculating Analog In Voltage shows how to calculate the voltage that is present on an analog input pin. When interfacing a device such as a thermometer, that represents temperature on one of its pins by voltage levels, a similar calculation will be needed to convert the value read on the analog pin to a temperature. The exact calculation depends on the characteristics of the thermometer.

Also refer to section 3.3.10 – AREF Pin for information on changing the voltage reference used by the analog pins or ADC.

3.3.4 TWI Bus Pins

TWI bus pins allow one or more TWI compatible device to be connected to an Arduino Uno. TWI devices share the SCL (Serial Clock) and SDA (Serial Data) pins. Each TWI device must have a unique TWI address which is set by the manufacturer of the device. Some devices allow the address to be changed by connecting certain of its pins to high or low logic levels (5V or GND voltage levels). The Arduino Uno acts as a master on the TWI bus and devices connected to it are slave devices that are controlled by the master. TWI is compatible with I²C devices and the I²C bus protocol.

Examples of TWI slave devices are EEPROM memory chips, I/O expansion chips, ADC (Analog to Digital Converter) and DAC (Digital to Analog Converter) devices, and temperature sensors or thermometers.

3.3.4.1 TWI Interfacing Example

Figure 3.15 shows two TWI devices connected to an Arduino Uno. When data is sent to a device on the TWI bus, or when data is requested from the TWI device, the address of the device is part of the request so that the correct device is accessed. The AT24C16C device shown in the figure is a TWI EEPROM chip with a fixed address. This means that only one of these devices can be connected to a TWI bus because all of these devices have the same address. Other TWI devices can be attached to the same TWI bus as long as they have different addresses from all other devices attached to the bus. The second device in the figure represents any other TWI device on the TWI bus. As can be seen in the figure, the SDA and SCL lines are shared between the two devices. The WP pin of the EEPROM chip is a write protect feature. This pin must be connected to GND in order to be able to write to the device. NC pins are pins that are not connected or unused.

Pull-up Resistors

Strictly speaking the TWI SDA and SCL lines should each have an external resistor of between 2k2 to 4k7 pulling them up to 5V. This is more important when more than one device is connected to the TWI bus and for longer connecting wires. The Wire library drivers enable the internal pull-up resistors on the TWI pins, which allows the TWI to work without external pull up resistors, but may make the TWI communications less reliable with some configurations. The configuration shown in Figure 3.15 with just the AT24C16C EEPROM connected using short wires works without external pull-up resistors.

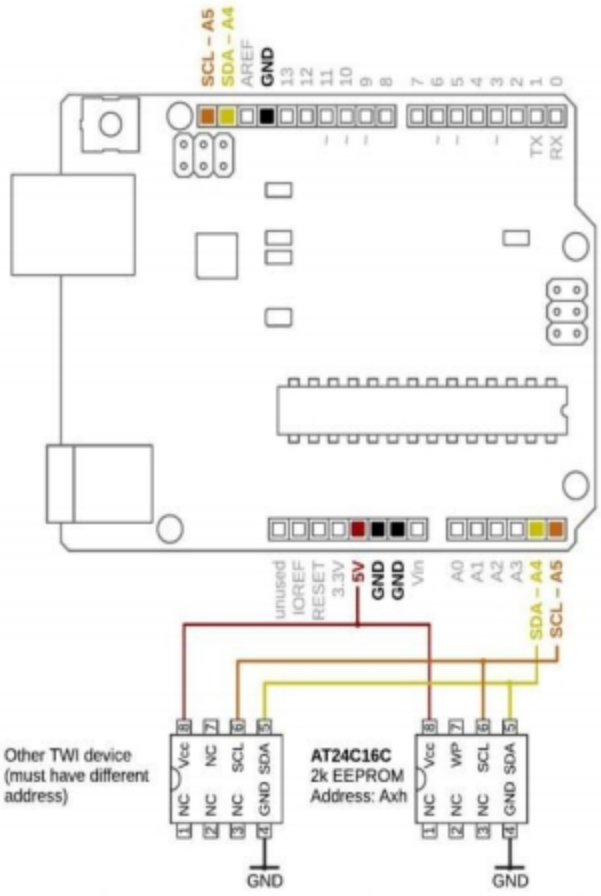


Figure 3.15: Arduino Uno Interfaced to Two TWI Devices

3.3.4.2 Accessing TWI Devices in Software

Use the Wire library in a sketch to access TWI bus devices. This means including the Wire.h header file in a sketch and using the Wire library object and functions found in the reference www.arduino.cc/en/Reference/Wire on the Arduino website. Find Wire example sketches in the Arduino IDE under File ▶ Examples ▶ Wire from the top menu.

The actual commands and sequence of bytes that must be sent in order to access a particular TWI device is found in the datasheet for the device and varies from device to device, although the actual TWI bus protocol is common to all devices. EEPROM devices need to receive their TWI address as well as the address of the byte in the EEPROM to access. Number of address bits and address byte order can be different for different devices, hence the necessity of referring to the correct datasheet. Note that the EEPROM Arduino library is for accessing the on-chip EEPROM of the ATmega328P microcontroller on Arduino Uno boards and not for accessing EEPROM chips on the TWI or SPI bus.

```
// Reads a single byte from a AT24C16C EEPROM on the TWI or I2C bus

#include <Wire.h>

void setup() {
  Wire.begin();         // join TWI bus as a master
  Serial.begin(9600);   // for displaying results in Serial Monitor window

  // *** Perform Random Read ***
  // address byte 0 of EEPROM
  Wire.beginTransmission((0xA0 >> 1));
  Wire.write(0x00);
  Wire.endTransmission();
  // read 1 byte from EEPROM at address 0
  Wire.requestFrom((0xA0 >> 1), 1);
  // *** End Perform Random Read ***

  while (Wire.available()) {          // check for response from EEPROM
    byte data8 = Wire.read();         // read byte from the EEPROM
    Serial.print("Byte read is: ");
    Serial.println(data8, HEX);       // display the byte
  }
}

void loop() {
}
```

In the above sketch, a single byte is read from address 0 of a AT24C16C EEPROM TWI device connected to the TWI pins of an Arduino Uno as shown in Figure 3.15. The value

read from EEPROM address 0 is sent out of the serial USB port and can be seen in the Serial Monitor window of the Arduino IDE. In a new EEPROM, this value is expected to be 0xFF, rather than 0x00. 0xFF is the default value of an erased byte in an EEPROM.
In the code from the above listing, the address of the EEPROM must first be sent and is 0xA0. This address includes 3 bits that are used to address a byte within the EEPROM. The next byte that is sent forms the lower 8 bits of the 11-bit address of the byte being accessed. After the EEPROM and byte have been addressed in this way, a single byte is requested from the device which is then read from the EEPROM and sent to the serial USB port. The address of the EEPROM is right shifted by 1, to convert it to a 7-bit address which is required by the Wire library. The sequence of sending the TWI device address and byte address to the EEPROM before reading the value of the addressed byte is obtained directly from the EEPROM datasheet, where it is called a random read – which means that this method can be used to address and read any byte in the EEPROM. In an EEPROM with a bigger storage capacity, two or more bytes may be used to address each data byte in the device. In the case of a two byte address, the EEPROM would be accessed by first sending its TWI address, followed by two bytes that make up the address of the data byte to access.

3.3.5 SPI Bus Pins

SPI bus pins allow one or more SPI device to be connected to the Arduino Uno. Devices connected to the SPI bus share the MOSI (Master Out Slave In), MISO (Master In Slave Out) and SCK (Serial Clock) pins. The Arduino Uno acts as the SPI bus master and controls the slave devices connected to the bus. Each slave device must have a separate slave select (SS) or chip select (CS) pin connected to a pin on the Arduino for addressing or selecting it. Although the SPI hardware in the ATmega328P microcontroller has one SS pin, it is not necessary to use this pin – any digital I/O pin can be used as the slave select pin. This pin is more useful if the Arduino were to be set up as a slave on the SPI bus as it would then be used by the SPI master to select the Arduino. An example of a shield that puts two devices on the SPI bus is the Ethernet shield that has an Ethernet chip and an SD card connected to the SPI bus. The Ethernet shield uses digital pin 4 for the chip select of the SD card, and digital pin 10 for the chip select of the Ethernet chip.

3.3.5.1 SPI Bus Interfacing Example

Figure 3.16 shows two micro SD card adapters connected to the common SPI bus pins and digital pin 10 used to select the adapter on the left. Digital pin 4 is used to select the adapter on the right. Note that these SD card adapters each have a circuit on-board that

translates voltage levels between the 5V Arduino Uno and 3.3V SD card. A 3.3V voltage regulator on the board steps down the 5V Arduino voltage to 3.3V to power the SD card.

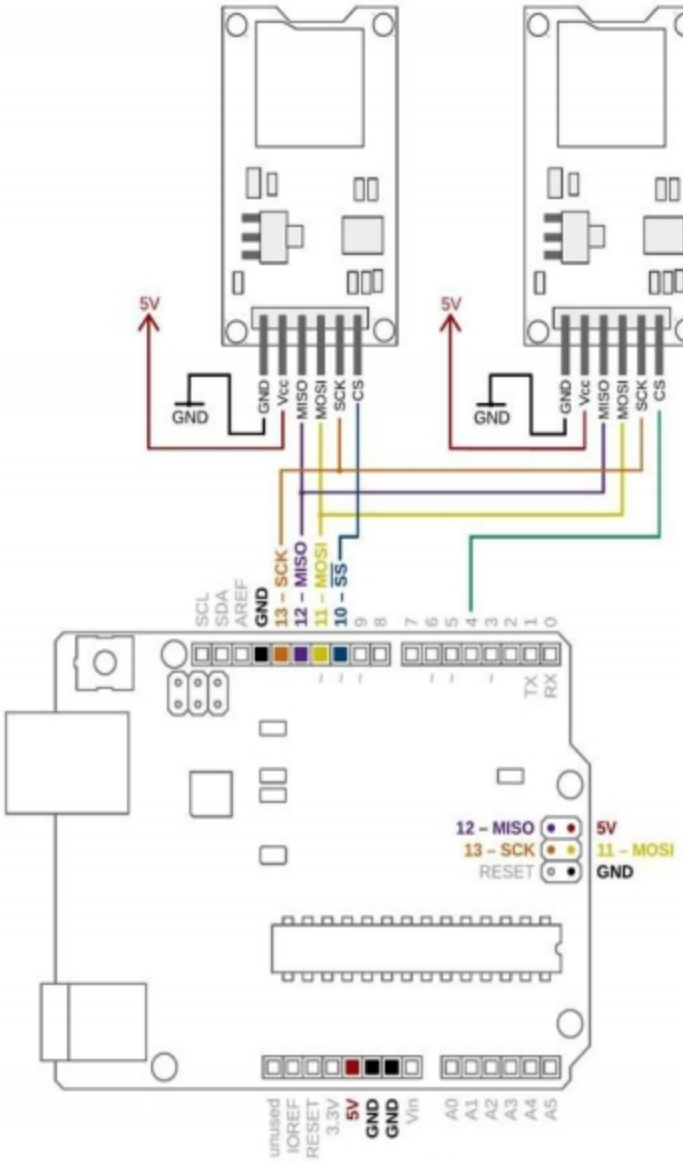


Figure 3.16: Arduino Uno Interfaced to Two SPI Devices

3.3.5.2 Accessing SPI Devices in Software

Use the Arduino SPI library to access SPI devices in a sketch. This means that the SPI.h header file must be included at the top of sketches that use the SPI library. Use the SPI object and functions from www.arduino.cc/en/Reference/SPI – the SPI reference page on the Arduino website.

SPI examples can be found in the Arduino IDE under File ► Examples ► SPI from the top menu. SD cards use the SD library which in turn uses the SPI library and is covered in the next section.

3.3.5.3 Accessing SD Cards in Software

The SD library is a layer of software that sits on top of the SPI drivers and handles access to the file system on the SD card. Go to www.arduino.cc/en/Reference/SD to find the SD card reference on the Arduino website. The library supports FAT16 and FAT32 file systems and short file names (8.3 format – file name of 8 characters maximum and file extension of 3 characters maximum, for example *filename.txt* is a name that is 8 letters long and has a 3 letter file extension).

SD card example sketches are very useful and can be found under File ► Examples ► SD from the top menu in the Arduino IDE. The CardInfo example is particularly useful when testing if a new SD card works with the Arduino Uno, as it displays information about the SD card in the Serial Monitor window of the Arduino IDE. The ReadWrite example writes a simple text file to the SD card, reads the contents of the file and displays it in the Serial Monitor window. Both of these examples use digital pin 4 as the chip select pin by default.

3.3.6 Serial / UART Pins

A serial port or Universal Asynchronous Receiver Transmitter (UART) sends and receives serial data asynchronously, which means without a separate square wave clock line for synchronization.

Although the ATmega328P microcontroller contains a USART (Universal Synchronous and Asynchronous Receiver Transmitter) in the form of internal USART hardware, this device is configured in UART mode on the Arduino Uno, so is referred to as a UART in this book. In the ATmega328P datasheet, information on this device can be found in the USART

section. The USART in the ATmega328P can be configured to operate in either synchronous mode (with a separate clock pin), or asynchronous mode. It is only ever used in an asynchronous configuration with the Arduino Uno.

Arduino Uno boards output serial data and receive data at 5V levels, also known as 5V TTL levels. To connect these pins to an RS-232 device, an interface chip is needed to convert the Arduino 5V levels to the RS-232 voltage levels which can be between ±3V to ±15V.

3.3.6.1 Hardware Serial Port

The Arduino Uno has one hardware serial port (which is a USART configured as a UART) that has the receive or RX line on digital pin 0, and the transmit or TX line on pin 1. These same lines connect to the ATmega16U2 serial to USB bridge via 1k resistors, which allows data sent and received on the serial port to be relayed across the USB connection. Hardware serial port means that the USART or UART hardware inside the ATmega328P microcontroller connects to these pins on the Arduino Uno.

If digital pins 0 and 1 are used as input or output pins, then the hardware serial port can not be used. If a serial device is connected to these pins, the communications between the connected serial device and the Arduino Uno will also appear on the USB port.

Go to www.arduino.cc/reference/en/language/functions/communication/serial/ for the serial port library reference. Serial example sketches can be found under File ► Examples ► 04.Communication from the top menu of the Arduino IDE.

3.3.6.2 Software Serial Port

Additional serial ports can be added to the Arduino Uno by using the SoftwareSerial library. Using the software serial library allows additional serial ports to be simulated in software by using the so-called “bit banging” method. From a hardware point of view this entails just selecting any digital pins for the serial transmit or receive pins and then configuring these pins in the SoftwareSerial driver in an Arduino sketch.

Go to www.arduino.cc/en/Reference/SoftwareSerial on the Arduino website for more information on the SoftwareSerial library. SoftwareSerial example sketches can be found under File ► Examples ► SoftwareSerial from the top menu of the Arduino IDE.

3.3.7 Power Pins

Power pins were already covered in section 2.7.1, however this section has a few more details on these pins. Figure 3.17 shows a block diagram of where the 5V, 3.3V and Vin voltages are derived from that are found on the header socket pins of an Arduino Uno.

3.3.7.1 GND Pins

GND is the common level of all of the power sources and is found on three of the socketed header pins down the sides of an Arduino Uno. GND is the negative or 0V connection of any power source.

3.3.7.2 5V Pin

5V DC found on the Arduino Uno 5V pin can be derived from either the USB 5V or from an external power supply connected to the power jack on the board, which is then stepped down to 5V by a regulator on the Arduino.

USB 5V

When powered from the USB connector, the circuitry on the Arduino Uno that makes up an automatic switch, as shown in Figure 3.17, switches the USB 5V onto the Arduino 5V pin. The same 5V is also fed into the 3.3V regulator. The voltage from a USB 2.0 port is specified as 5V ± 5% and up to 500mA current.

External Power to 5V Regulator

With an external power supply of 7 to 12V DC voltage plugged into the external power jack on an Arduino Uno, the 5V on the Arduino 5V pin is derived from this external power source after it is fed through a 5V regulator. The automatic switch circuitry shown in the block diagram of Figure 3.17 will block the USB 5V if external power is present.

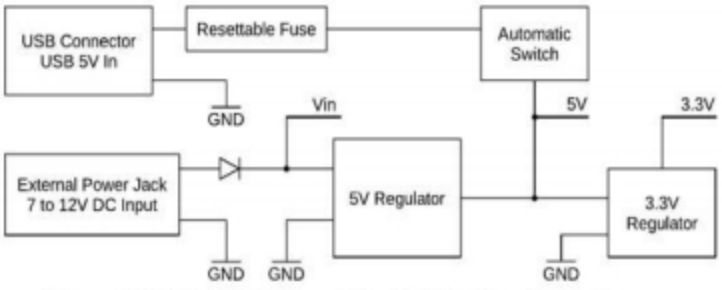


Figure 3.17: Block Diagram of the Arduino Uno Power Sources

The 5V regulator is able to deliver an output current of up to 1A (one amp), however the more current that is drawn through the regulator, the hotter the regulator will get. The 5V regulator has an **absolute maximum** input voltage of 20V. The higher the external input voltage, the more power will be dissipated in the 5V regulator, and the hotter it will get. Be aware that a fully charged 12V lead acid battery has a voltage of 13.8V. The main restriction on the amount of current that can be drawn from the regulator is thermal because this regulator does not have adequate heat sinking on the Arduino Uno.

3.3.7.3 3.3V Pin

3.3V DC found on the Arduino 3.3V pin is derived from a 3.3V regulator that is fed with 5V from either USB, through the automatic switch (Figure 3.17), or the 5V regulator when the Arduino Uno is externally powered. The regulator, and therefore pin can deliver up to 150mA.

3.3.7.4 Vin Pin

The Vin pin voltage is derived from an external power supply connected to the power jack of the Arduino Uno. This external DC voltage is fed through a diode before arriving at the Vin pin (See Figure 3.17), which means that the voltage on the Vin pin will be around 0.7 to 0.8V lower than the external power supply voltage because of the forward voltage drop across the diode. Also see section 2.4.4 for information on battery powering the Arduino Uno through this pin. Vin is available to any shield that is plugged into the Arduino Uno.

3.3.8 Reset Pin

A reset pin is found on the same connector that the power pins are located on, on an Arduino Uno. Reset is active low, which means that a low signal must be applied to the reset pin to reset the ATmega328P microcontroller. On an Arduino Uno, the reset pin is pulled high via a 10k resistor, which means that an external normally open reset switch can be connected to the reset pin and GND as shown in Figure 3.18, if an external reset switch is needed. When the switch is open, the reset pin is pulled to 5V via the 10k resistor on the Arduino, keeping it out of reset. If the switch is momentarily closed, the reset pin is pulled low to GND, resetting the microcontroller. When the switch is released, the 10k resistor keeps the microcontroller out of reset again, allowing it to start running the bootloader and then the currently loaded sketch. The reset pin resets only the main microcontroller on the Arduino Uno and not the ATmega16U2. This means that the Arduino will not be taken off the USB bus if it is reset using either the reset pin or the on-board reset button, when it is plugged into a host computer via USB.

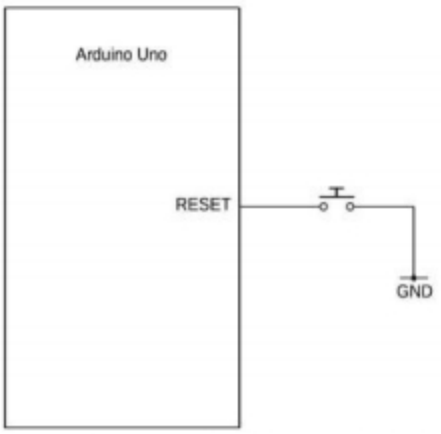


Figure 3.18: Connecting an External Reset Switch to an Arduino Uno

Active low pin labels normally have a line above them to show that they are active low, which means RESET would be written $\overline{\text{RESET}}$, but this is not done on the Arduino Uno reset label.

3.3.9 IOREF Pin

The IOREF pin (input/output reference) is connected directly to the Arduino Uno 5V pin. IOREF is a standard pin where a shield plugged into an Arduino board can detect the voltage that the board operates at so that it can switch in the correct voltage level translator circuits for the board. In this way the same shield can be used for 5V and 3.3V Arduino boards.

3.3.10 AREF Pin

The AREF pin allows an alternate reference voltage to be used with the ADC converter for reading analog input voltages on pins A0 to A5. Analog voltages on A0 to A5 are limited to a maximum voltage that is equal to the reference voltage. By default the reference voltage is 5V taken from the 5V supply. If a more stable or different reference voltage is required, it can be applied to the AREF pin. Figure 3.19 shows an example of a 3V reference applied to the AREF pin of an Arduino Uno. The 3V reference voltage is derived from a REF3030 voltage reference chip which steps down an input voltage of 3.3V to 3V. This is just one example of a voltage reference chip. There are many other parts available that provide different reference voltages for a range of different input voltages.

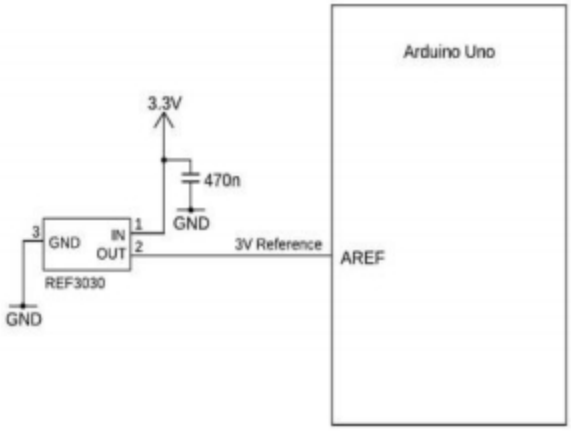


Figure 3.19: Arduino Uno with an External ADC Reference Voltage

When using an external reference voltage applied to the AREF pin, analogReference() must be called before reading an analog value on A0 to A5. Call this function as follows.

```
analogReference(EXTERNAL);
```

Cross refer to section 2.7.6 for basic information on the AREF pin.

3.4 ICSP Header on Main Microcontroller

Figure 3.20 shows the pinout of the ICSP header for the main microcontroller found at the end of the Arduino Uno board, on the opposite side to the USB connector and external power jack. The ICSP header is used to program the ATmega328P microcontroller using an external programmer. It is also used by some shields, such as the Ethernet shield to connect to these pins. Programmer documentation and documentation for AVR micro-controllers that use this type of programming header usually refer to it as the ISP (In-System Programming) header. Only on Arduino boards is it referred to as the ICSP header.

Notice that all of the ICSP header pins are available on the Arduino Uno socket headers at the sides of the board. If an external circuit is connected to these pins and an external programmer is used on the ICSP header, the circuit may interfere with the programmer.

Cross refer to section 1.1.4.11 – ICSP Header, for basic information on the ICSP header, and section 2.8 – Programming Headers, for an example of connecting an external programmer to the ICSP header. Section 2.9.4 – ICSP SPI Pins and Reset, shows the ICSP header names and connections to the Arduino pins at the edge of the Arduino Uno board. These same ICSP header pins used to program the ATmega328P are also the SPI pins – cross refer to 3.1.2 – Shared SPI Pins.

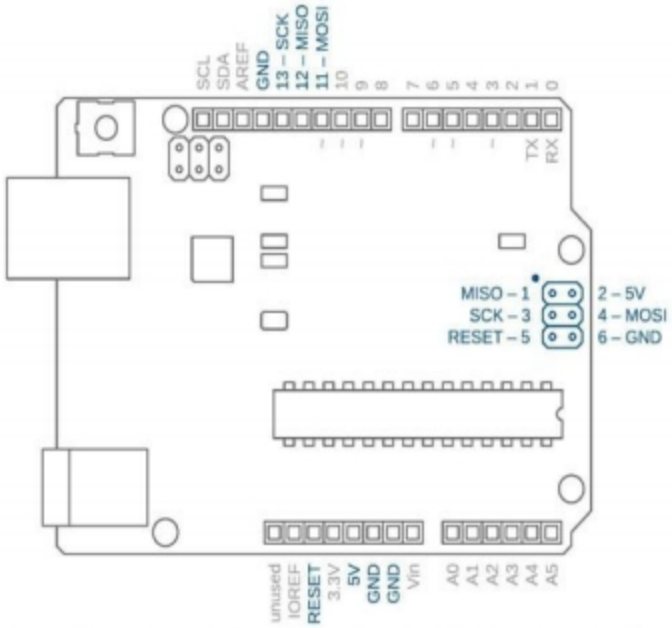


Figure 3.20: ICSP Header Pins and Connections to Arduino Pins

3.5 ICSP Header on USB Microcontroller

Figure 3.21 shows the pinout for the ICSP header that connects to the ATmega16U2 USB microcontroller. Although this header has the same pinout and names as the main ICSP header, it is separate from the main ICSP header. The pins of this header connect to the ATmega16U2 only. In the Arduino Uno schematic these pins are labeled MISO2, MOSI2, SCK2 and RESET2 to distinguish them from the same pins on the main ICSP header, and the ICSP header is labeled ICSP1.

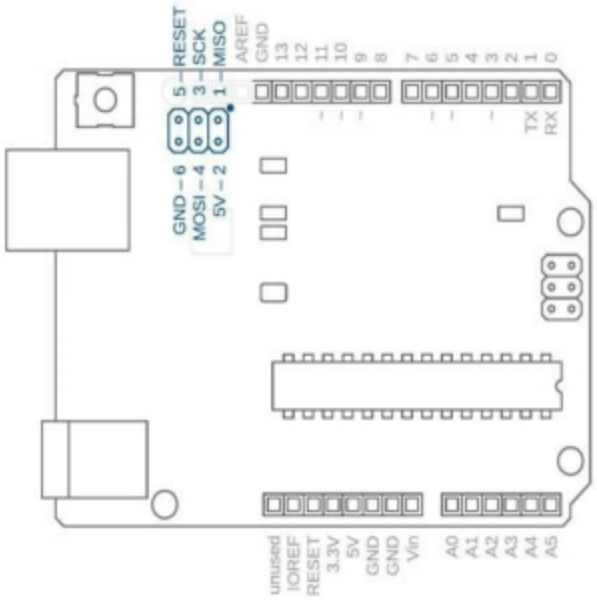


Figure 3.21: ICSP Header Pinout for the ATmega16U2 USB Microcontroller

3.6 JP2 Header on USB Microcontroller

Four pins connect from the ATmega16U2 USB microcontroller to the JP2 header solder pads as shown in Figure 3.22 on the next page. Arduino have not given any explanation for these pins or solder pads, and it is assumed that they have been taken to the pads so that advanced users who are programming the ATmega16U2 can easily access these pins for convenience for whatever application they want.

Table 3.1 on the next page shows the pin names of the JP2 header and the alternate functions of these pins. Use of these pins is for advanced users who must refer to the ATmega16U2 datasheet for the explanation of these alternate functions.

In the Arduino Uno schematic, JP2 is inverted, or flipped over as if it were to be placed on the bottom layer of the board, but on the board itself, JP2 is correctly oriented for the top layer of the board. This may cause some confusion when looking at the schematic and then Figure 3.22, but if the Eagle CAD file for the board is opened, it can be clearly seen that Figure 3.22 has the correct pin names for JP2.

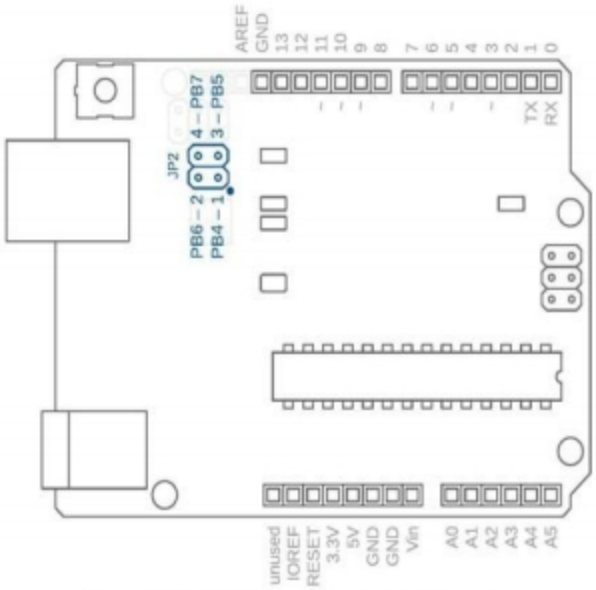


Figure 3.22: JP2 Pinout on the Arduino Uno

Table 3.1: JP2 Pins and Alternate Functions

JP2 Header Pin	ATmega16U2 Port Pin	Alternate Pin Functions
1	PB4	T1 / PCINT4
2	PB6	PCINT6
3	PB5	PCINT5
4	PB7	PCINT7 / OC0A / OC1C

3.7 Finding the Datasheets

Anyone wanting to find out more about the microcontrollers on an Arduino Uno board can refer to the datasheets for these devices. Datasheets are essential for advanced users who are programming the hardware directly, for example in plain C language, or for developers writing driver libraries for the Arduino.

3.7.1 ATmega328P Datasheet

Find the ATmega328P datasheet at www.microchip.com/wwwproducts/en/ATmega328p on the Microchip website. Find the datasheet as well as application notes and other documentation under the Documents tab on the page.

3.7.2 ATmega16U2 Datasheet

Go to www.microchip.com/wwwproducts/en/ATmega16u2 to find the datasheet for the ATmega16U2 microcontroller used as the USB to serial bridge chip on Arduino Uno boards.

3.7.3 Datasheets for Other Components

Datasheets for other components used on Arduino Uno boards can easily be found by searching the internet. Refer to chapter 6 where the part numbers for components can be found in the circuit diagram and parts list. Simply enter the part number into a search engine to find the datasheet for the part.

Chapter 4 • Power Reference

In this power reference chapter for the Arduino Uno, the power supplies from both USB and external sources are explained. Specifications for external and USB power are included.

Power supply circuits on the Arduino Uno board are simplified by breaking them up into sections in the circuit diagram. Each section is then explained.

In this Chapter

- Power sources and specifications
- On-board power supply and regulator circuit diagram
- Power supply protection

4.1 Power Supply Specification

This section summarizes the power supply specification of an Arduino Uno in the categories that follow.

4.1.1 Operating Voltage

The Arduino Uno operates at 5V which is derived either from the USB 5V or from an external power supply that is dropped to 5V by the on-board regulator. The on-board 5V regulator is a 1.0A regulator with its output voltage trimmed to ±1.0%.

In addition to the 5V regulator on the Arduino Uno, a 3.3V regulator drops the 5V supply to 3.3V which is available on the Arduino 3.3V pin. This regulator has an output voltage tolerance of 1.5%. The 3.3V regulator can deliver a maximum current of 150mA.

Cross refer to section 3.3.7 for information on the Arduino Uno power pins.

4.1.2 USB Power Input

USB power from a USB 2.0 port is specified as 5V ± 5% and up to 500mA current. The USB 5V is protected by a resettable fuse that will disconnect the USB 5V should more than 500mA be drawn from the USB port. This fuse has a small resistance which means that as more current is drawn from the USB port, the voltage drop across the fuse will increase. The actual voltage supplied to the Arduino will therefore be the USB voltage minus this small volt drop.

Cross refer to section 2.4.1 and 2.4.2 for information on USB power and USB cable specification.

4.1.3 External Power Jack Input

An external power input of 7 to 12V DC can be supplied to the power jack input that is found on the same end of the board as the USB connector. Arduino specify the limits of this input as 6 to 20V and warn that a voltage less then 7V may result in the 5V regulator supplying less than 5V which may make the board unstable. A voltage more than 12V can cause the 5V regulator to overheat and damage the board. See the next section for the pinout and polarity of the external power jack.

4.1.4 External Power Jack Pinout

When an external supply of 7 to 12V DC is supplied to the power jack, the center pin of the jack must be connected to positive of the power supply. The external power jack is a 2.1mm barrel connector. Figure 4.1 shows the pinout of the power connector as well as a typical symbol from a power supply that indicates it is wired with the center pin positive.

Cross refer to section 2.4.3 for information on external power and section 2.4.4 for information on battery powering the Arduino Uno.

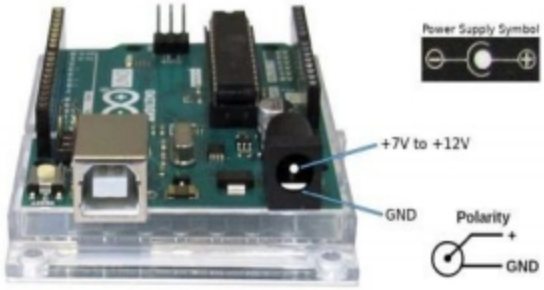


Figure 4.1: External Power Jack Pinout and Polarity

4.1.5 Vin Pin as Power Input

Vin, found on the power header socket of the Arduino Uno, can be supplied with 7 to 12V DC. The Arduino documentation states that a battery can be connected between Vin and GND to power the Arduino Uno from a battery. Positive of the battery must connect to the Vin pin. Cross refer to section 2.4.4 for information on battery powering the Arduino Uno.

4.2 Power Circuit

Figure 4.2 on the next page shows the power supply circuit of the Arduino Uno broken up into parts to make it simpler to understand. Each part of the power supply circuit is explained in the sections that follow.

4.2.1 External Power In and 5V Regulator

Connector X1 seen at the top left of Figure 4.2 is the 2.1mm barrel connector jack for the external power supply. The external power is fed to U1, the 5V regulator, through diode

D1. This 5V regulator only functions when external power is supplied to the Arduino Uno. When powered from USB alone, this regulator is not used.

4.2.2 Power On Indicator LED

The green ON LED seen at the top right of Figure 4.2 switches on when it receives 5V from either the 5V regulator or from USB. The ON LED has an equivalent current limiting series resistor value of 500Ω, made up from two 1k resistors in parallel.

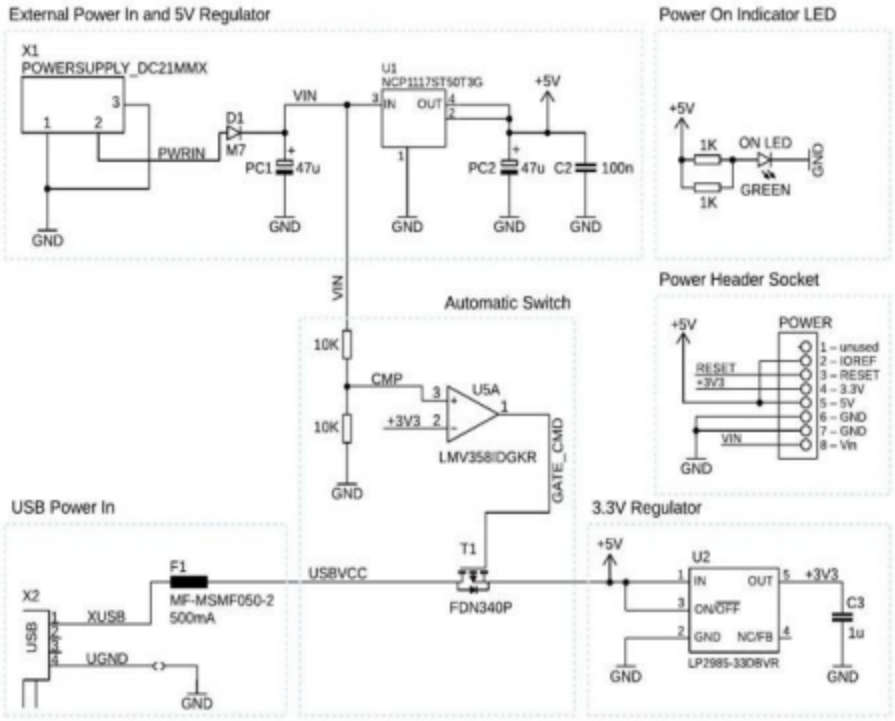


Figure 4.2: Arduino Uno Power Supply Circuit

4.2.3 USB Power In

X2, seen at the bottom left of Figure 4.2, is the USB connector on the Arduino Uno. It is shown here without the USB data lines connected. USB +5V input is connected to resettable fuse F1. USB GND (UGND in the figure) connects to GND via a normally closed link consisting of two solder pads and a connecting track. This allows the track to be broken to separate the USB GND from the main GND on the board. The pads allow it to be soldered closed again. This connector is found under the Arduino Uno near the pins of the USB connector. It is more difficult to see on older Arduino Uno REV3 boards

because it is covered in white silkscreen. On newer REV3 boards, the pads of this link are exposed and clearly visible.

4.2.4 Automatic Switch

An automatic switch detects whether external power is supplied to the Arduino Uno, and if it is, disconnects the USB 5V supply. If external power is not detected, USB power is switched onto the +5V points shown in Figure 4.2. The automatic switch consists of operational amplifier U5A working as a comparator. If no external power is connected to the Arduino, and it is powered from USB alone, assume that the output of the op-amp is at GND, switching the P-channel MOSFET T1 on which supplies USB 5V to the circuit. The + pin of the op-amp will then be at a voltage of approximately 2.25V, and the – pin at a voltage of +3.3V. This causes the op-amp to keep transistor T1 on which keeps the USB +5V connected to the Arduino +5V connections. The 2.25V on the + pin of the op-amp is derived from 5V from the USB fed back through the internal protection diode of regulator U1. This 5V loses about 0.5V from the regulator protection diode forward voltage drop, and appears on VIN in the circuit diagram. It can be measured on the Vin pin as around 4.5V. The two 10k resistors on the + input of the op-amp act as a voltage divider, dividing the 4.5V by two to get around 2.25V.

If external power is supplied to the Arduino Uno, then the two 10k resistors acting as a voltage divider put half VIN onto the + pin of the op-amp, which is higher than the +3.3V on the – pin. This causes the op-amp to switch transistor T1 off, disconnecting the USB positive voltage from the rest of the circuit. +5V from the 5V regulator then supplies 5V to the Arduino Uno.

4.2.5 3.3V Regulator

The 3.3V regulator simply produces +3.3V from +5V that is supplied to it. This +5V will either be from the USB or from the 5V regulator if powered externally.

4.2.6 Power Header Socket

+5V, +3.3V, VIN and GND are all connected to pins on the Arduino Uno POWER header socket as can be seen in Figure 4.2. At the right of the connector in the figure are the pin names found silkscreened on the Arduino Uno board (except for pin 1, unused, which is not marked on the board). +5V in the circuit becomes 5V on the connector, +3.3V becomes 3.3V, VIN becomes Vin and GND remains GND. These are just slightly different names for the same things found in the Arduino Uno schematic and on the Arduino Uno board.

4.3 Power Supply Protection

The power supply circuit shown in Figure 4.2 has some protection against reverse polarity connection, current overload and short circuit as described in the sections that follow.

4.3.1 Reverse Polarity Protection

Diode D1, as seen in Figure 4.2, offers some basic reverse polarity protection to power supplied to the external power jack. If external power is supplied to the Arduino Uno via the Vin pin, then this diode is bypassed and no reverse polarity protection is available. This diode also prevents any voltage being fed back onto the power jack when the Arduino Uno is powered from USB alone.

4.3.2 5V Regulator Protection Features

5V regulator U1 offers output current limiting and thermal shutdown. This device has an unlimited output short circuit duration, which means that shorting its output will not destroy the device because internal protection circuits protect it from output short circuits. An internal thermal limiting circuit protects this regulator against excessive temperature. If the device gets too hot, the output will switch off until it cools down to a suitable temperature.

4.3.3 3.3V Regulator Protection Features

U2, the 3.3V regulator, has over-current and thermal protection. This device also has internal short circuit protection that prevents the device from being destroyed by a short circuit on its output.

4.3.4 USB Overload Protection

Resettable fuse F1 is a 500mA fuse that protects the USB 5V from overload and short circuit. The fuse will automatically reset when the overload or short circuit is removed.

Chapter 5 • Arduino Uno Firmware and Bootloader

Firmware is present on both the main Arduino Uno microcontroller and the USB to serial bridge microcontroller. This firmware is factory loaded when the Arduino Uno is manufactured.

Firmware on the main microcontroller is known as the bootloader and allows software sketches to be loaded to the Arduino from the Arduino IDE via the USB connection.

The USB to serial bridge microcontroller on the Arduino Uno has DFU bootloader firmware, and firmware that gives it the USB to serial bridge functionality. The DFU bootloader allows the USB to serial firmware to be updated using the USB connection without the need for an external programmer.

This chapter explains all of the firmware found on the Arduino Uno.

In this Chapter

- Updating the USB to serial firmware using DFU
- Using Microchip Studio to back up and load new firmware
- How to reload the USB to serial bridge microcontroller firmware using ICSP
- Loading the main microcontroller bootloader to a new ATmega328P chip
- Microcontroller fuse settings
- Purpose of the RESET-EN solder jumper

5.1 Updating the USB to Serial Bridge Firmware using DFU

The most basic task that an Arduino user can do with the ATmega16U2 microcontroller is to update its operational firmware using the DFU (Device Firmware Update) method. This means that the firmware or software that gives this device its USB to serial bridge function can be updated using the USB port on the Arduino Uno. Most Arduino users should never need to do this, however it is included in this manual to make it complete. Advanced users may also want to use the DFU programming method to put their own custom software into the ATmega16U2. DFU programming requires no external programmer or additional hardware device. Some additional software will need to be downloaded to do the programming.

For Windows computers, download the FLIP (Flexible In-system Programmer) software from Microchip at www.microchip.com/developmenttools/ProductDetails/FLIP
For additional information on Arduino DFU programming and for DFU programming using Linux and Mac, see www.arduino.cc/en/Hacking/DFUProgramming8U2 on the Arduino website. A brief description of DFU programming an Arduino Uno using the Microchip FLIP software on a Windows computer follows.

Before updating the ATmega16U2 firmware, download and install the FLIP software, as mentioned in the previous paragraph. Connect the Arduino Uno to a USB port of the computer, as would be done to upload a sketch. Connect the RESET pin of the ATmega16U2 ICSP header to GND and then disconnect it in order to reset the ATmega16U2. Make sure to connect to the RESET signal on the ICSP header near the USB connector and not the one at the end of the board. Refer to Figure 3.21 in section 3.5 of Chapter 3 for the pinout of the ICSP header. The RESET pin of the header can be connected to GND on the header, or on one of the GND pins of the header sockets at the edges of the board.

After resetting the ATmega16U2, start the FLIP software. Click the chip icon or select Device ▶ Select... from the top menu of the FLIP software. In the dialog box that pops up, scroll down to find ATmega16U2 and click it to select it. Click the OK button in the dialog box to accept. Click the USB icon and then USB on the menu that pops up, or select Settings ▶ Communication ▶ USB from the top menu of the FLIP application. Choose File ▶ Load HEX File from the top menu. Use the dialog box that opens to navigate to the firmware file for the ATmega16U2. This can be found in the Arduino IDE folder in **arduino-1.8.13\hardware\arduino\avr\firmwares\atmegaxxu2\arduino-**

usbserial which can easily be found if the zipped Arduino IDE software was downloaded and unzipped to the Windows Desktop as described in section 1.1.5 of Chapter 1. The base folder of the above path will be different for different versions of the Arduino IDE software. Find the file **Arduino-usbserial-atmega16u2-Uno-Rev3.hex** in the **arduino-usbserial** folder from the above path, select it in the FLIP dialog box, and then click the OK button. Finally click the Run button in the main FLIP application window. Erase, Program and Verify should be selected by default above this button. The Flash memory in the ATmega16U2 will be erased and programmed with the selected HEX file. After the Flash memory has been programmed, unplug the USB cable from the Arduino Uno and then plug it back in. If everything went well, the new ATmega16U2 firmware will start running and the Arduino will appear in Windows as a COM port device as would normally be expected. The board can now be loaded with sketches using the Arduino IDE software as usual.

5.2 Microchip Studio

Microchip Studio, which was rebranded from Atmel Studio in late 2020, is a full IDE with C and C++ development tools that can be used to develop software for AVR and other microcontrollers that were originally manufactured by Atmel, but now by Microchip. Microchip Studio contains a device programming utility that can be used not only for loading firmware to both the ATmega16U2 and ATmega328P microcontrollers in the form of HEX or BIN files, but can also be used to set the fuses on these devices. Fuses are used to set the start-up state of some of the microcontroller settings such as the initial clock frequency as well as to provide write protection to some of the Flash memory segments.

Arduino users who want to program the bootloader and set the correct fuse settings on a new ATmega328P microcontroller when replacing this device have the option of using the Arduino IDE, as described in section 2.8.2 of Chapter 2, or to use Microchip Studio and its device programming utility. There are other methods of programming both the Flash and fuses of new microcontrollers such as using command line applications in Linux and Windows (usually using the *avrdude* utility), but using Microchip Studio is far easier. To summarize, Microchip Studio is available only for Windows, and requires a USB programming device such as the Atmel-ICE to program microcontrollers through the ICSP header. A command line utility is also available for programming AVR microcontrollers and can be used with a variety of USB or other programming devices, including home-made programmers. Alternatively use the Arduino IDE (section 2.8.2).

Those Arduino users who want to load the bootloader to the ATmega328P microcontroller and set its fuses, program the ATmega328P directly using C or C++, or even program the ATmega16U2 to load both the DFU firmware and operational firmware, may want to install Microchip Studio. Remember that a USB programming device that connects to the ICSP header is also required when using Microchip Studio. Table 5.1 lists some of the AVR USB programming devices from Microchip that work with Microchip Studio. The Atmel-ICE replaces the AVRISP mkII and AVR Dragon which are no longer available for sale, but still work with Microchip Studio.

Table 5.1: Microchip USB Programmers for AVR Microcontrollers

USB Programmer	Part Number	Web Page
Atmel-ICE	atatmel-ice	www.microchip.com/developmenttools/ProductDetails/atatmel-ice
AVRISP mkII (no longer available)	atavrisp2	www.microchip.com/developmenttools/ProductDetails/atavrisp2
AVR Dragon (no longer available)	ATAVRDRAGON	www.microchip.com/Developmenttools/ProductDetails/ATAVRDRAGON

To install Microchip Studio, go to www.microchip.com/en-us/development-tools-tools-and-software/microchip-studio-for-avr-and-sam-devices on the Microchip website. At the time of writing, Microchip Studio version 7 was the current version. Microchip Studio version numbering continues from the last version of Atmel Studio, so there are no Microchip Studio versions 1 to 6. Use either the web installer or offline installer. The web installer requires a small file to be downloaded and run, which will then download and install the selected Microchip Studio components. The offline installer is a big file that can be downloaded and run to do the Microchip Studio installation on one or more computers without needing to connect to the internet or download anything else while installing.

Microchip Studio and a USB programming device are used in the sections that follow for backing up the firmware from the ATmega16U2 and ATmega328P microcontrollers, programming new firmware to these devices and for setting their fuses. Connectors found on the USB programmers in Table 5.1 are actually ISP headers or connectors and not ICSP, but connect to the ICSP headers on Arduino Uno boards. In the text that follows the name ICSP is used instead of ISP to avoid confusion.

5.3 USB Microcontroller Firmware

There are two firmware applications that are present in the Flash memory of the USB microcontroller (ATmega16U2) on an Arduino Uno. First, the DFU firmware is a bootloader that allows the operational software of the ATmega16U2 to be updated using the USB connection of the Arduino, as discussed in section 5.1 of this chapter. The operational software that gives the ATmega16U2 its USB to serial bridge functionality is the second firmware program found in the Flash memory of this microcontroller.

Arduino Uno boards prior to revision 3 were fitted with an ATmega8U2. Table 5.2 shows the memory sizes of the ATmega8U2 and ATmega16U2. This manual refers to the ATmega16U2 as the USB microcontroller chip that is found on all new Arduino Uno boards, rather than the ATmega8U2.

Table 5.2: Memory Sizes for the ATmega16U2 and ATmega8U2

Device	Flash Memory	SRAM	EEPROM
ATmega16U2 (Arduino Uno REV3)	16k	512 bytes	512 bytes
ATmega8U2 (Arduino Uno REV1 and REV2)	8k	512 bytes	512 bytes

5.3.1 Backing up the ATmega16U2 Firmware with Microchip Studio

As a precaution, the firmware on the ATmega16U2 can be copied from this device on the Arduino Uno board and saved to HEX and BIN files using the device programming utility found in Microchip Studio. In this way, if anything goes wrong when loading new firmware to the ATmega16U2, the original firmware can be restored. Microchip Studio must be installed on the computer used to read the firmware from the microcontroller. Additionally, a USB programming device will be needed, such as one that is listed in Table 5.1.

When plugging the ICSP connector from the programmer into the ICSP header for the ATmega16U2 on the Arduino Uno, the key on the connector prevents the connector from being pushed into the header because it bumps up against the header socket at the edge

of the board. The key on the connector refers to the small raised plastic key in the middle of the connector which prevents the connector from being inserted into a shrouded header the wrong way around. A shrouded header has a slot that the key fits into. A solution to this problem is to use short wires that have a male pin on one end and female socket on the other end. Plug the male end of 6 of these wires into the connector on the programmer and the female ends into the corresponding pins of the ATmega16U2 ICSP header on the Arduino Uno. Refer to Figure 3.21 in section 3.5 of Chapter 3 which shows the header and which pin is pin 1, that must line up with pin 1 on the programming cable header. As few users will need to program the ATmega16U2, rather than use up space in this book with an image of the connection, refer to wspublishing.net, the supporting website for an image of the connection.

Plug the programmer, such as the Atmel-ICE, into the USB port of the computer. Plug the ICSP connector of the programmer into the ICSP header near the USB connector, as described in the previous paragraph, making sure that it is connected the right way around. Now power the Arduino Uno board, either via a USB cable, or using an external power supply.

Start Microchip Studio and click the Device Programming icon on the top toolbar, or select Tools ► Device Programming from the top menu. In the Device Programming dialog box that pops up, select the programmer being used under Tool, e.g. Atmel-ICE, if this is the tool being used. Under Device, select ATmega16U2 for an Arduino Uno REV3 board, or ATmega8U2 for a REV2 or REV1 board. Make sure that the Interface is set to ISP. Click the Apply button. To check that the programmer and target device have been set up correctly and that Microchip Studio can “see” the target device, click the Read button next to Device signature. If all is set up correctly, the device signature and target voltage will be read from the ATmega16U2.

In the Device Programming dialog box, click Memories in the left column. Click the Read... button in the Flash section of the dialog box. Use the Save As dialog box to navigate to the folder to save the Flash contents to. In the same dialog box, give the target file a name in the File name: field. The file type can be changed from the default Intel Hex (.hex) type to binary (.bin) if desired, but the default is fine. Click the Save button when ready. The device programming utility will read the contents of the ATmega16U2 Flash memory and save it to the selected file. In Windows, use the File Explorer to navigate to the destination folder that was chosen. The HEX or BIN file should

be found in this folder. The file, whether HEX or BIN format, contains the entire Flash memory contents, and not just the firmware found in the Flash. This means that it also contains the blank bytes from the Flash, which make the file bigger than the original HEX file that was used to factory load the DFU and operational firmware. The HEX or BIN file can now be used to restore the entire Flash memory contents of the ATmega16U2 if ever needed. Programming this file to memory can also be done using the device programming utility from Microchip Studio as described in section 5.3.4 that follows in this chapter.

5.3.2 DFU Bootloader Firmware

C source code for the DFU bootloader firmware can be found in the Arduino IDE folder **arduino-1.8.13\hardware\arduino\avr\firmwares\atmegaxxu2\arduino-usbdfu** where the base folder of this path will be different for different versions of the Arduino IDE. Go to github.com/arduino/ArduinoCore-avr/tree/master/firmwares/atmegaxxu2/arduino-usbdfu, which is a copy of the same source code at GitHub.

To actually restore the DFU firmware, together with the USB to serial bridge firmware, go to **arduino-1.8.13\hardware\arduino\avr\firmwares\atmegaxxu2** in the Arduino IDE folder, which contains the combined HEX file that has both of these firmware programs in one file. The file **Arduino-COMBINED-dfu-usbserial-atmega16u2-Uno-Rev3.hex** in this folder can be loaded to an Arduino Uno to restore both the DFU firmware and USB to serial bridge firmware at the same time. Refer to section 5.3.4 that follows for information on using Microchip Studio to program a HEX file to the ATmega16U2.

The DFU bootloader firmware gives the ATmega16U2 its ability to load new operational software to its Flash memory using the DFU method described in section 5.1 of this chapter.

5.3.3 USB to Serial Bridge Firmware

C source code and the HEX file for the USB to serial bridge firmware can be found in the Arduino IDE folder **arduino-1.8.13\hardware\arduino\avr\firmwares\atmegaxxu2\arduino-usbserial** which contains the HEX file for the USB to serial bridge firmware on its own. This is the file used in section 5.1 of this chapter that explains how to load this file to the Arduino Uno using the DFU method. It is available here uncombined with the DFU firmware so that the DFU method can be used to update the USB to serial bridge firmware using a USB connection. The same source code and HEX files can be found on GitHub at

github.com/arduino/ArduinoCore-avr/tree/master/firmwares/atmegaxxu2/arduino-usbserial which contains the latest version of this code and HEX files.

The USB to serial bridge firmware sets up the ATmega16U2 as a USB device that relays serial data between the main Arduino Uno microcontroller (ATmega328P) and the computer connected to the Arduino Uno USB port.

5.3.4 Programming the USB Microcontroller using ICSP

Connect a USB programmer to the ICSP header for the ATmega16U2 found near the USB connector as described in section 5.3.1 of this chapter. In fact connect to the Arduino Uno using a USB programmer and Microchip Studio exactly as described in section 5.3.1 – the only difference is that the ATmega16U2 will be programmed instead of reading the contents of its Flash memory.

In Microchip Studio, open the Device Programming utility by clicking the Device Programming icon, or clicking Tools ► Device Programming from the top menu. In the dialog box that pops up, select the correct USB programmer under Tools, select ATmega16U2 under Device, select ISP under Interface and then click the Apply button. Click the Read button under Device signature to make sure that the settings are correct and that the Device Programming utility can connect to the target microcontroller. If all of the settings are correct, the device signature will be retrieved and target voltage read and displayed in the dialog box (only after clicking the Read button).

To load a HEX file to the ATmega16U2, click Memories in the left column of the Device Programming dialog box. Click the browse for file button [...] in the Flash section of the dialog box to open a dialog box that will be used to navigate to the desired HEX file. The HEX file can be any HEX file that is made to run on the ATmega16U2 microcontroller of the Arduino Uno. This could be the HEX file from section 5.3.1 of this chapter that was retrieved from the ATmega16U2, or combined DFU firmware file described in section 5.3.2. Navigate to the desired HEX file using the dialog box and then click the Open button. Click the Program button in the Flash section of the Device Programming dialog box to load the selected HEX file to the ATmega16U2.

After loading the new firmware to the ATmega16U2, power down the Arduino Uno and then unplug the USB programmer from it. Power the Arduino again and it should work as expected – verify by loading a sketch to the Arduino from the Arduino IDE.

5.3.5. ATmega16U2 Fuse Settings

In order to see the fuse settings of the ATmega16U2 or to change them, connect to the ATmega16U2 using a USB programmer exactly as described in the previous section, section 5.3.4. Open the Device Programmer in Microchip Studio as described, but instead of Flash in the left column of this dialog box, click Fuses. Most Arduino users should never need to change the fuse settings for the ATmega16U2. The default Arduino Uno settings for the ATmega16U2 fuses are include in Table 5.3 below in case they ever need to be set back to their correct settings.

Table 5.3: ATmega16U2 Fuse Settings for the Arduino Uno

Fuse Name	Value
EXTENDED.BODLEVEL	Brown-out detection level at VCC=3.0V
EXTENDED.HWBE	☑
HIGH.DWEN	unchecked
HIGH.RSTDISBL	unchecked
HIGH.SPIEN	☑
HIGH.WDTON	unchecked
HIGH.EESAVE	unchecked
HIGH.BOOTSZ	Boot Flash size=2048 words start address=$1800
HIGH.BOOTRST	unchecked
LOW.CKDIV8	unchecked
LOW.CKOUT	unchecked
LOW.SUT_CKSEL	Ext.Crystal Osc. 8.0- MHz; Start-up time: 16K CK + 4.1ms

When the fuses are set as shown in Table 5.3, the fuse registers will have the following values, as seen at the bottom of the Fuses section in the Device Programming dialog box.

EXTENDED	0xF4
HIGH	0xD9
LOW	0xEF

More information on each fuse and what the fuse settings do can be found in the ATmega16U2 datasheet.

5.4 Main Microcontroller Bootloader

If the main microcontroller (the ATmega328P) is ever replaced on an Arduino Uno and does not have a bootloader already programmed into it, the bootloader will need to be loaded to this device. This can be done using Microchip Studio and an external USB programmer that connects to the ICSP header. Programming is done in the same way as has already been described for the ATmega16U2, but the ICSP header at the end of the board, on the opposite side to the USB connector, must be used. In the same way, the bootloader can be backed up from an ATmega328P that already contains a bootloader. These actions are explained further in the sections that follow and include the fuse settings for the ATmega328P, when used on an Arduino Uno. The bootloader can also be quickly restored from within the Arduino IDE as has already been described in section 2.8.2, but with Microchip Studio installed, this will not work because of a driver clash.

5.4.1 Backing up the ATmega328P Firmware with Microchip Studio

As a precaution, the firmware on the ATmega328P can be copied from this device on the Arduino Uno board and saved to HEX and BIN files using the device programming utility found in Microchip Studio. This is especially appropriate when using a clone board that may have a different bootloader than the one used on genuine Arduino Uno boards.

Plug a USB programmer, such as one listed in Table 5.1, into the computer and then plug the cable with the 6 pin socket connector from the programmer into the ICSP header at the end of the Arduino Uno board. Refer to Figure 3.20 in section 3.4 for the pinout of the ICSP header for the ATmega328P which shows where pin 1 is. Figure 2.9 in Chapter 2 shows the correct orientation of a Atmel-ICE programmer plugged into the ICSP header of an Arduino Uno – the raised plastic key on the connector points towards the USB connector. Power up the Arduino Uno using either USB or an external power supply.

Start Microchip Studio and open the Device Programming utility by either clicking the Device Programming icon on the top toolbar, or by selecting Tools ▶ Device Programming from the top Microchip Studio menu. In the Device Programming dialog box, select the correct USB programmer under Tools, select ATmega328P under Device, make sure that ISP is selected under Interface, finally click the Apply button. To check that everything is set up correctly and that Microchip Studio can connect to the target device, click the Read button under Device signature. If all is well, the device signature and target voltage will be read and displayed in the Device Programming dialog box.

In the left column of the Device Programming dialog box, click Memories. In the Flash section of Memories in the dialog box, click the Read… button to navigate to a folder to save the HEX file to that will be retrieved from the Flash memory of the ATmega328P. At the bottom of the Save As dialog box enter the name of the HEX file that will be saved – this can be any valid name of your choice. Click the Save button when done. The Device Programming utility will read the contents of the ATmega328P Flash memory and save it to a HEX file of the chosen name. This HEX file can be used to restore the original firmware to the ATmega328P at any time. The HEX file includes the entire contents of the Flash memory including whatever user sketch was programmed in the Flash at the time of creating the HEX file.

5.4.2 Optiboot Bootloader Firmware

Arduino Uno boards now use the Optiboot bootloader in the ATmega328P microcontroller which replaces an older bootloader. Optiboot uses a 512 byte segment of Flash memory, whereas the older bootloader used up 2k of Flash memory. This means that an extra 1.5k of Flash memory is available for user sketches.

Optiboot C source code and HEX files can be found in the Arduino IDE folder in the path **arduino-1.8.13\hardware\arduino\avr\bootloaders\optiboot** where the base folder will have a different name for different versions of the Arduino IDE. The same files can be found at github.com/arduino/ArduinoCore-avr/tree/master/bootloaders/optiboot on GitHub. The Optiboot project on GitHub is at github.com/Optiboot/optiboot and the Optiboot wiki can be found at github.com/Optiboot/optiboot/wiki for more information on Optiboot.

Optiboot flashes the L LED of the Arduino when it starts. It then either starts running the currently loaded sketch, or waits for a new sketch to be loaded if reset by the Arduino IDE. See github.com/Optiboot/optiboot/wiki/HowOptibootWorks for more details.

5.4.3 Restoring the Bootloader

The Optiboot bootloader can be restored to the ATmega328P, or loaded to a new blank ATmega328P microcontroller, using Microchip Studio and an external USB programmer, as has already been described in this chapter for the ATmega16U2. Use the same method for connecting to the ATmega328P from Microchip Studio that was used for backing up the bootloader from section 5.4.1. See section 7.2 in chapter 7 for information on replacing the ATmega328P microcontroller on an Arduino Uno board.

To load the Optiboot bootloader to the ATmega328P on the Arduino Uno, connect the USB programmer to the ICSP header at the end of the board as is described in section 5.4.1 earlier in this chapter. Open the Device Programming utility in Microchip Studio, and connect to the programmer and ATmega328P in the same way. Click the Memories item in the left column of the Device Programming dialog box. Click the [...] button in the Flash section of the dialog box to open a dialog box that is used to navigate to the HEX file to load to the ATmega328P. The HEX file for restoring the bootloader can either be the HEX file created when backing up the bootloader, as described in section 5.4.1, or the correct HEX file from **arduino-1.8.13\hardware\arduino\avr\bootloaders\optiboot** which for the Arduino Uno is **optiboot_atmega328.hex**. Select the correct HEX file in the dialog box and then click the Open button. Back in the Device Programming dialog box, click the Program button in the Flash section of the dialog box. If everything is set up correctly, the Flash will be programmed with the selected HEX file.

If the bootloader was loaded to a new blank ATmega328P, the fuses must be set correctly in this device before continuing, as described in the next section. If the fuses are already set correctly, close the Device Programming dialog box in Microchip Studio and unplug the power from the Arduino Uno. Disconnect the USB programmer from the Arduino and then power up the Arduino again. If the fuses are already set correctly, then test the newly loaded bootloader by opening the Arduino IDE and loading a test sketch to the Arduino using the normal USB cable method.

5.4.4 ATmega328P Fuse Settings

A new blank ATmega328P must first have the Optiboot bootloader loaded to it as described in the previous section. ATmega328P fuses must then be set correctly for use in the Arduino Uno. Fuse values can be read and set using the Device Programming utility in Microchip Studio. To read and set the fuses, first connect an external USB programmer

to the ICSP header at the end of the Arduino Uno as is described in section 5.4.1 earlier in this chapter. In Microchip Studio, open the Device Programming utility, choose the correct settings for the tool, device and interface as already described in section 5.4.1 and then click the Apply button to connect. In the left column of the Device Programming dialog box, click the Fuses item.

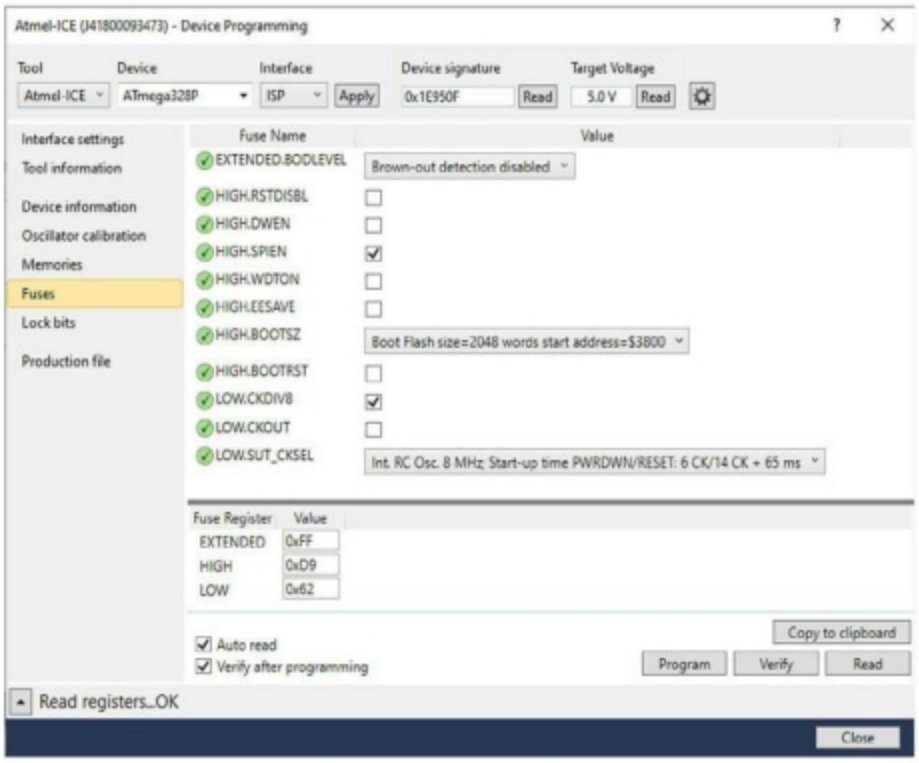


Figure 5.1: Default Fuse Settings for a New Blank ATmega328P

Figure 5.1 shows what to expect when looking at the fuse values of a new blank ATmega328P microcontroller in the Device Programming dialog box. Default fuse register values are as follows.

EXTENDED	0xFF
HIGH	0xD9
LOW	0x62

These fuses must be programmed to the values shown in Figure 5.2 and Table 5.4. The following fuses must be changed from their default values:

EXTENDED.BODLEVEL to Brown-out detection at VCC=2.7V
HIGH.EESAVE to checked (optional, see next page for explanation)
HIGH.BOOTSZ to Boot Flash size=256 words start address=$3F00
HIGH.BOOTRST to checked
LOW.CKDIV8 to unchecked
LOW.SUT_CKSEL to Ext. Crystal Osc. 8.0- MHz; Start up time PWRDWN/RESET: 16K/14 CK + 65ms

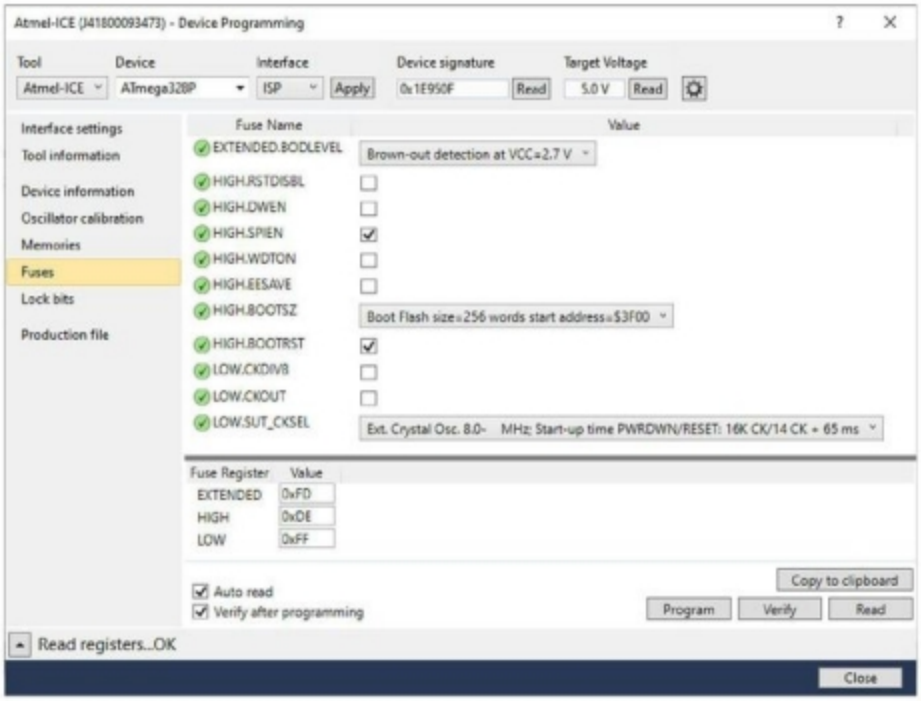


Figure 5.2: ATmega328P Fuse Settings for the Arduino Uno

When these settings have been made, the fuse register values at the bottom of the dialog box will be:

EXTENDED	0xFD
HIGH	0xDE (EESAVE unchecked) or 0xD6 (EESAVE checked)
LOW	0xFF

It was found that EESAVE was checked on older Arduino Uno R3 boards, but unchecked on new Arduino Uno R3 Boards. If the EESAVE box (HIGH.EESAVE fuse) is checked in the dialog box, it protects the EEPROM memory contents when the chip is erased.

Table 5.4: Arduino Uno ATmega328P Fuse Settings

Fuse Name	Value
EXTENDED.BODLEVEL	Brown-out detection level at VCC=2.7V
HIGH.RSTDISBL	unchecked
HIGH.DWEN	unchecked
HIGH.SPIEN	☑
HIGH.WDTON	unchecked
HIGH.EESAVE	unchecked on new boards, see above paragraph
HIGH.BOOTSZ	Boot Flash size=256 words start address=$3F00
HIGH.BOOTRST	☑
LOW.CKDIV8	unchecked
LOW.CKOUT	unchecked
LOW.SUT_CKSEL	Ext.Crystal Osc. 8.0- MHz; Start-up PWRDWN/RESET: 16K CK/14 CK + 65 ms

After making the changes, click the Program button near the bottom of the dialog box. This will program the new fuse values to the ATmega328P. Click OK in the warning dialog box to continue. Close the Device Programming dialog box, unplug the power from the Arduino Uno and then unplug the USB programmer. Plug the Arduino Uno into the computer USB port and then load a sketch to it from the Arduino IDE to make sure that the bootloader is working and that the fuse settings are correct.

5.5 The RESET-EN Solder Jumper

Some of the AVR USB programmers have debugging capability, namely, the AVR Dragon (which is no longer being sold by Microchip) and the Atmel-ICE (which replaces older USB programmers). Note that the AVRISP mkII (which is also no longer sold by Microchip) does not have any debugging capabilities as it is a programmer only. This also applies to the many hobby programmers that are available, such as the USBasp and USBtinyISP, which are both program-only devices. Debugging capabilities refers to the ability of the USB programmer / debugger to work in conjunction with software such as Microchip Studio, and allow the user, or programmer, to access the main microcontroller on the Arduino in order to view the contents of its memory and internal registers, as well as to single-step through source code and insert breakpoints. This capability is not available in the Arduino IDE version 1.8.13 which is the current version at the time of writing.

In order to use the debugging capabilities of the USB programmer / debugger, it is plugged into the ICSP header as normally would be done for programming the Flash memory of the target AVR device, or for reading fuse values. A C or C++ program can then be loaded to the AVR using Microchip Studio. Microchip Studio can then be used to debug the target AVR.

The debugging system on the ATmega328P microcontroller is known as debugWIRE and allows the debugging software to connect to the target microcontroller through the ICSP header using a USB programmer that has debugging capabilities. In order for debugWIRE to work on the ATmega328P on Arduino Uno boards, reset circuitry must be disconnected from the reset pin. Arduino Uno boards have a solder link labeled RESET-EN that consists of two solder pads connected by a circuit board track. The track between the solder pads must be cut through in order to disconnect part of the reset circuitry from the reset pin when using debugWIRE. The connection can be remade by soldering the two solder pads together. Figure 5.3 shows the location of the RESET-EN solder pads on the Arduino Uno.

Again, this feature will only be used by advanced Arduino users who are typically programming the board using plain C and Microchip Studio. Those readers interested in learning the C language to program Arduino boards may be interested in the book *C Programming with Arduino ISBN 978-1-907920-46-2* by the same author and published by Elektor. This book contains a chapter on debugging Arduino Uno and MEGA 2560 boards using Atmel Studio, which has since been replaced with Microchip Studio.

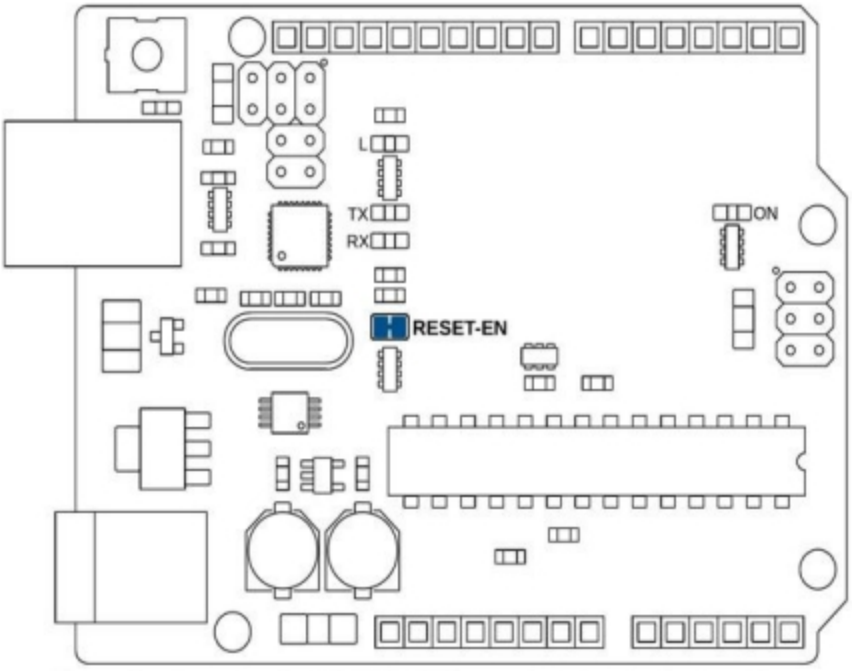


Figure 5.3: Location of the RESET-EN Solder Jumper on an Arduino Uno

5.6 Alternative Firmware Programming Methods

Alternative methods for programming the bootloader to an Arduino Uno can be found at playground.arduino.cc/Main/ArduinoCoreHardware/#Bootloader at the Arduino playground under the Bootloader section. This includes information on using a second Arduino as a programmer to program a target Arduino as well as how to use alternate programmers. Using an Arduino as an ISP (In-system programmer) is also described on the Arduino website at www.arduino.cc/en/Tutorial/BuiltInExamples/ArduinoISP

See Chapter 2, section 2.8 for more information on loading sketches to the Arduino Uno using an external programmer and the Arduino IDE. Refer to the same section for information on restoring the bootloader to the main microcontroller, also using an external programmer and the Arduino IDE.

Refer to www.arduino.cc/en/Hacking/DFUProgramming8U2 for additional information on DFU programming the ATmega16U2.

Chapter 6 • Circuit Diagram and Components

Although the circuit diagram for the Arduino Uno is available online in both PDF and Eagle EDA formats, this chapter breaks the circuit diagram up into easier to reference and understand sections.

This chapter also includes the component list of all of the parts found on the Arduino Uno board and in the schematic, as well as the location of parts on the board itself.

Parts are usually difficult to locate on the board, especially the smaller parts such as resistor packs and capacitors, because the board is compact and there is no space to silkscreen reference designators for the parts. The diagram in this chapter that shows the location of all components on the board is a great reference that can be used to easily locate any component.

In this Chapter

- Arduino Uno block diagram
- Arduino Uno circuit diagram split into three easy to reference figures
- Component list for the Arduino Uno
- Location of parts on the board

6.1 Circuit Diagram

The circuit diagram for REV 3 Arduino Uno boards follows in this chapter, and is also shown as a block diagram that consists of three main parts, namely, the power supply circuit, the USB microcontroller circuit and the main microcontroller circuit with header sockets. Each of these three parts are then presented as separate circuits in the sections that follow.

6.1.1 Block Diagram

The Arduino Uno block diagram, shown in Figure 6.1, is a top level view of the Arduino Uno circuit diagram that shows how the different parts of the circuit interconnect. The power supply circuit is supplied with power from the 2.1mm jack and from the USB 5V. This circuit then supplies 5V to the rest of the Arduino and 3.3V to the 3.3V pin. The USB microcontroller circuit connects to the main microcontroller circuit via the TX and RX serial port lines and reset line via C5. This allows the Arduino IDE to reset the main ATmega328P microcontroller in order to start its bootloader which then allows a new sketch to be loaded to the main microcontroller using the TX and RX lines. GND is not shown in the block diagram, but is common throughout. Vin is also omitted in the figure, but connects from the 2.1mm jack through a diode to a pin on the header sockets.

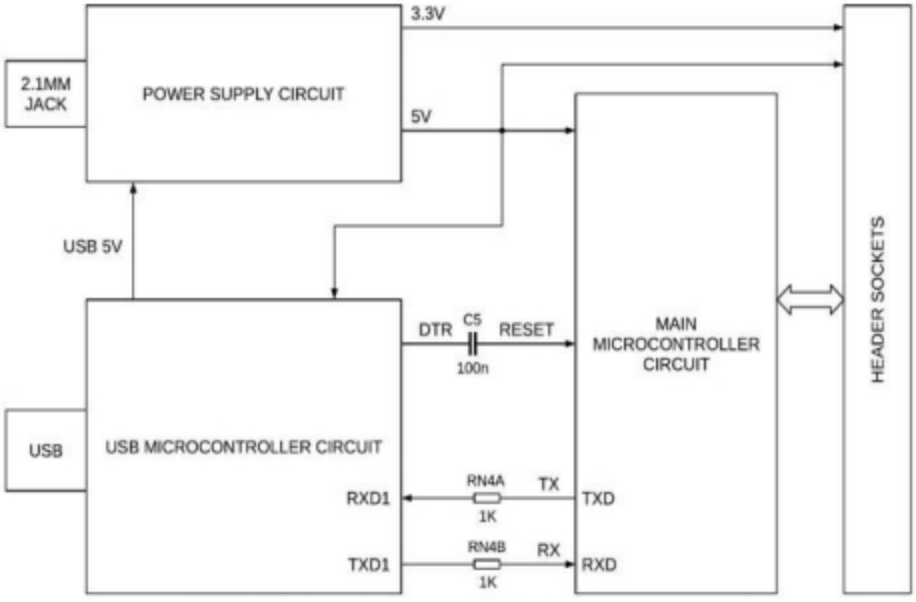


Figure 6.1: Block Diagram of Arduino Uno Schematic or Circuit Diagram

6.1.2 Main Microcontroller Circuit

Figure 6.2 shows the main microcontroller circuit containing the ATmega328P device. Something to note in the circuit diagram is the unusual names for reference designators, such as RESET for the reset switch, where one would expect a reference designator such as SW1, or similar, for a switch. The same for the L LED – L is the actual reference designator rather than LED1, D1, or similar. Header socket connectors have reference designators of IOH, AD and IOL. All of these unusual reference designators are found in the component list in section 6.2 of this chapter. Arduino pin names have been included next to the header sockets at the right of the circuit diagram, as they appear on the silk-screen of the Arduino board. Double arrows at the left show connections to Figure 6.3.

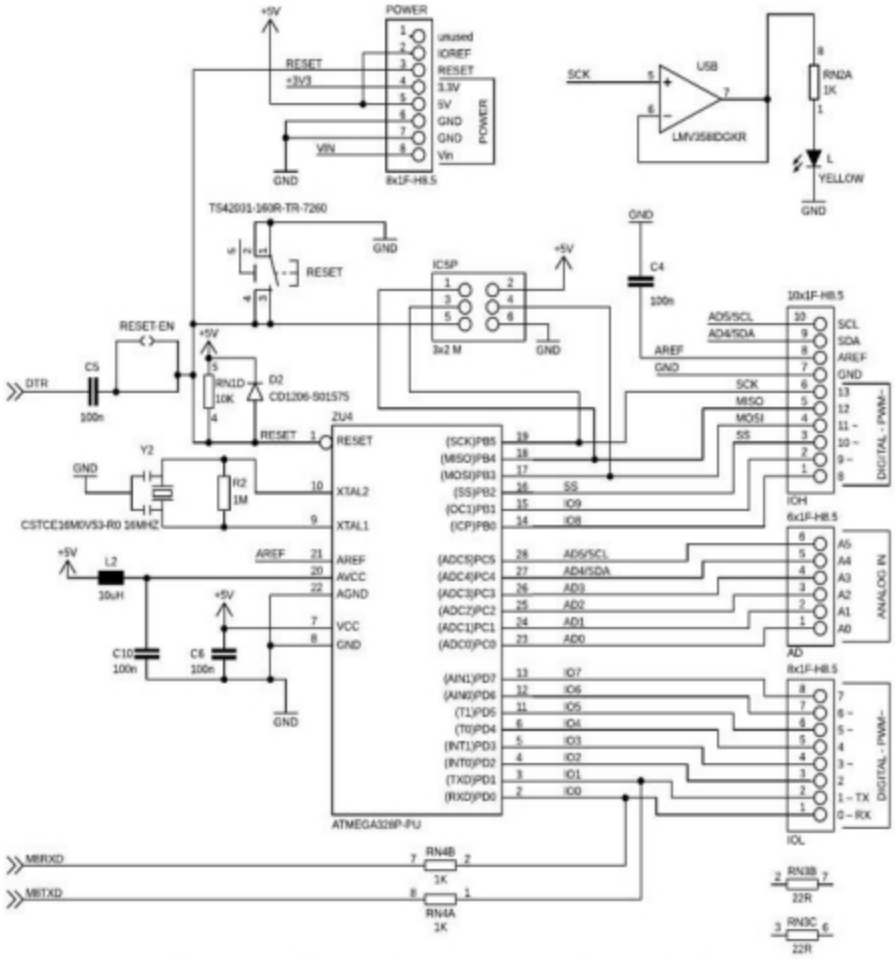


Figure 6.2: Arduino Uno Main Microcontroller Circuit

6.1.3 USB Microcontroller Circuit

The USB microcontroller circuit containing the ATmega16U2 device can be seen in Figure 6.3. The figure has been rotated to make it bigger on the page.

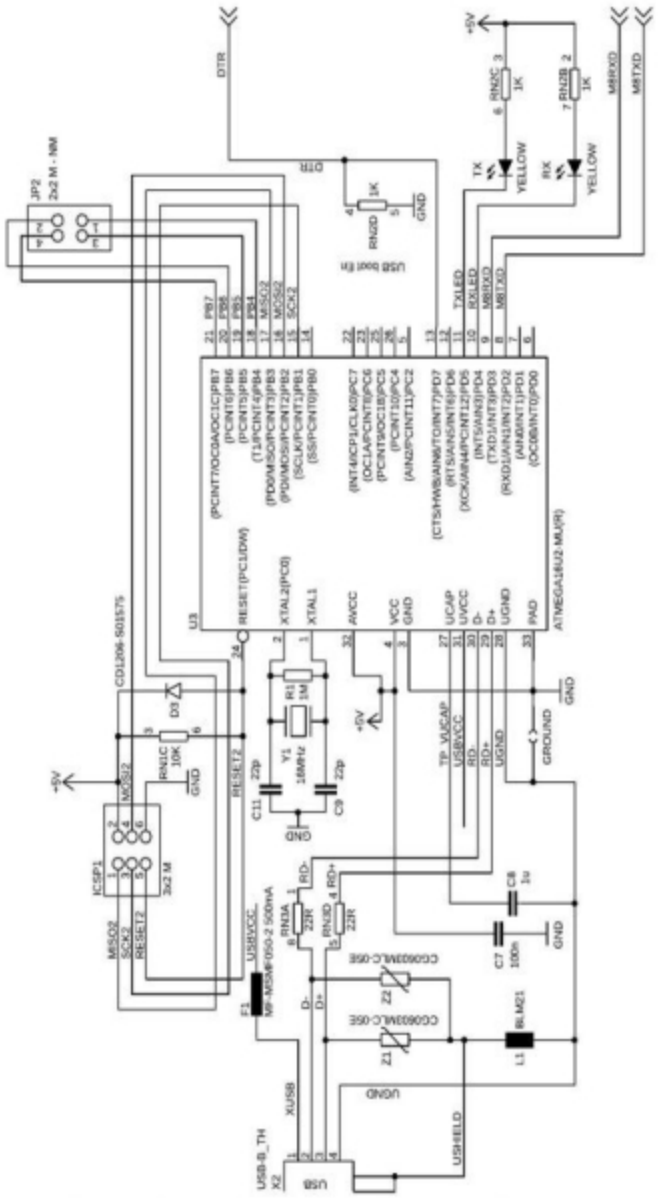


Figure 6.3: Arduino Uno USB Microcontroller Circuit

6.1.4 Power Supply Circuit

Figure 6.4 shows the power supply circuit of the Arduino Uno. Cross refer to Figure 4.2 in Chapter 4 which shows the same circuit but includes the USB connector and power header socket. USBVCC connects to fuse F1 from the Arduino Uno USB microcontroller circuit in Figure 6.3. Refer to chapter 4 for details of the power supply circuit. Note that U5 is a dual op-amp chip that contains two op-amps, U5A which is used as the comparator in the power supply circuit and U5B which is used as a unity gain buffer for the L LED in Figure 6.2. It is powered on pins 8 and 4 as can be seen in Figure 6.4

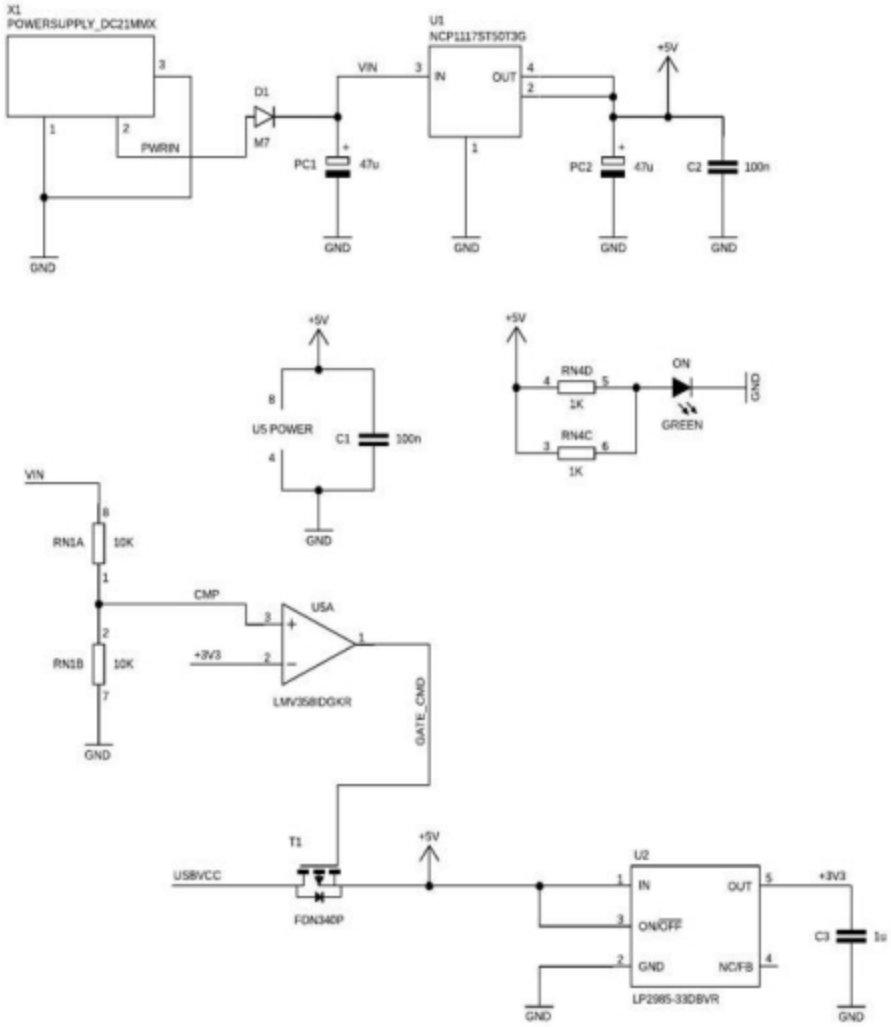


Figure 6.4: Arduino Uno Power Supply Circuit

6.2 Component List

Table 6.1 contains the component list for all of the components found on the Arduino Uno. The table extends over a few pages and lists the components in alphabetical order by reference designator. The device and package columns use the names provided from the circuit via Eagle EDA and may not always make sense. Read the comments column for a better description of the part.

Table 6.1: Arduino Uno Component List

Reference Designator	Value	Device	Package	Comments
AD	6x1F-H8.5	PINHD-1X6	1X06	ANALOG IN header socket containing A0 to A5 pins.
C1	100n	C-EU0603-RND	C0603-ROUND	Capacitor
C2	100n	C-EU0603-RND	C0603-ROUND	Capacitor
C3	1u	C-EU0603-RND	C0603-ROUND	Capacitor
C4	100n	C-EU0603-RND	C0603-ROUND	Capacitor
C5	100n	C-EU0603-RND	C0603-ROUND	Capacitor
C6	100n	C-EU0603-RND	C0603-ROUND	Capacitor
C7	100n	C-EU0603-RND	C0603-ROUND	Capacitor
C8	1u	C-EU0603-RND	C0603-ROUND	Capacitor
C9	22p	C-EU0603-RND	C0603-ROUND	Capacitor
C10	100n	C-EU0603-RND	C0603-ROUND	Capacitor
C11	22p	C-EU0603-RND	C0603-ROUND	Capacitor
D1	M7	DIODE-SMB	SMB	Diode
D2	CD1206-S01575	DIODE-MINIMELF	MINIMELF	Diode
D3	CD1206-S01575	DIODE-MINIMELF	MINIMELF	Diode
F1	MF-MSMF050-2 500mA	L-EUL1812	L1812	500mA resettable fuse.

Table 6.1: Arduino Uno Component List continued...

Reference Designator	Value	Device	Package	Comments
GROUND		SJ	SJ	Solder jumper. Part of PCB copper on bottom PCB layer.
ICSP	3x2 M	PINHD-2X3	2X03	Main microcontroller ICSP header.
ICSP1	3x2 M	PINHD-2X3	2X03	USB microcontroller ICSP header.
IOH	10x1F-H8.5	PINHD-1X10_ARD	1X10	Header socket containing Arduino pins 8 to 13.
IOL	8x1F-H8.5	PINHD-1X8	1X08	Header socket containing Arduino pins 0 to 7.
JP2	2x2 M - NM	PINHD-2X2	2X02	4 pin solder pads wired to ATmega16U2.
L	YELLOW	LEDCHIP-LED0805	CHIP-LED0805	L LED on pin 13.
L1	BLM21	WE-CBF_0805	0805	Inductor – ferrite bead
L2	10uH	0715_CV201210-100K	0805	Inductor
ON	GREEN	LEDCHIP-LED0805	CHIP-LED0805	Green ON LED power indicator.
PC1	47u	CPOL-EUD	PANASONIC_D	Electrolytic capacitor.
PC2	47u	CPOL-EUD	PANASONIC_D	Electrolytic capacitor.
POWER	8x1F-H8.5	PINHD-1X8	1X08	POWER header socket.
R1	1M	R-EU_R0603	R0603-ROUND	Resistor
R2	1M	R-EU_R0603	R0603-ROUND	Resistor
RESET	TS42031-160R-TR-7260	TS42	TS42	Reset push button switch.

Table 6.1: Arduino Uno Component List continued...

Reference Designator	Value	Device	Package	Comments
RESET-EN		SJ	SJ	RESET-EN solder jumper on top copper layer.
RN1	10K	4R-NCAY16	CAY16	4 resistor pack.
RN2	1K	4R-NCAY16	CAY16	4 resistor pack.
RN3	22R	4R-NCAY16	CAY16	4 resistor pack.
RN4	1K	4R-NCAY16	CAY16	4 resistor pack.
RX	YELLOW	LEDCHIP-LED0805	CHIP-LED0805	RX LED.
T1	FDN340P	PMOSSOT23	SOT-23	P-channel MOSFET (transistor).
TX	YELLOW	LEDCHIP-LED0805	CHIP-LED0805	TX LED.
U1	NCP1117ST50T3G	MC33269ST-3.3T3	SOT223	5V regulator.
U2	LP2985-33DBVR	LP2985-XXDBVR33	SOT23-DBV	3.3V regulator.
U3	ATMEGA16U2-MU(R)	ATMEGA16U2-MU	MLF32	ATmega16U2 microcontroller.
U5	LMV358IDGKR	LMV358MMX	MSOP08	Dual op-amp.
X1	POWERSUPPLY_DC21MMX	POWERSUPPLY_DC21MM	POWERSUPPLY_DC-21MM	2.1mm barrel connector.
X2	USB-B_TH	PN61729	PN61729	USB B-type connector.
Y1	16MHz	XTAL/S	QS	16MHz crystal.
Y2	CSTCE16M0V53-R0 16MHZ	RESONATORMU	RESONATOR	16MHz ceramic resonator.

Table 6.1: Arduino Uno Component List continued...

Reference Designator	Value	Device	Package	Comments
Z1	CG0603MLC-05E	VARISTORCN0603	CT/CN0603	Protection varistor for USB data line.
Z2	CG0603MLC-05E	VARISTORCN0603	CT/CN0603	Protection varistor for USB data line.
ZU4	ATMEGA328P-PU	ATMEGA328P-PU	DIL28-3	Main ATmega328P microcontroller.

Note that there is no U4 part in the circuit or component list, but rather ZU4 is the ATmega328P microcontroller reference designator. C10 and L2 were added to newer Arduino Uno REV3 boards which are referred to as Rev3e boards in the schematic and build files. On older REV3 boards these two components are not present. They form a power supply filter for the analog power pin AVCC of the ATmega328P microcontroller. This pin supplies power to the ADC in the ATmega328P.

GROUND and RESET-EN are both normally closed solder pad jumpers that are made out of copper PCB track on the board. The connecting track between the pads on these components can be cut with a sharp blade to open circuit these components. Solder can be used to join the pads back together again if they are ever cut.

6.3 Component Positions on the Board

Figure 6.5 on the next page shows the positions of each of the components on the Arduino Uno board. The corresponding components can be found in the circuit diagram of figures 6.2, 6.3 and 6.4, as well as in the component list of Table 6.1.

Note that the GROUND solder jumper is on the bottom layer of the board and can be seen between C8 and Z1 in Figure 6.5, just behind the USB connector X2. On older revision 3 Arduino Uno boards, C6 is in the position that C10 is in the figure. Older boards do not have a C10 capacitor, in other words on older boards, there is no capacitor where C6 is, it is where C10 is in the figure.

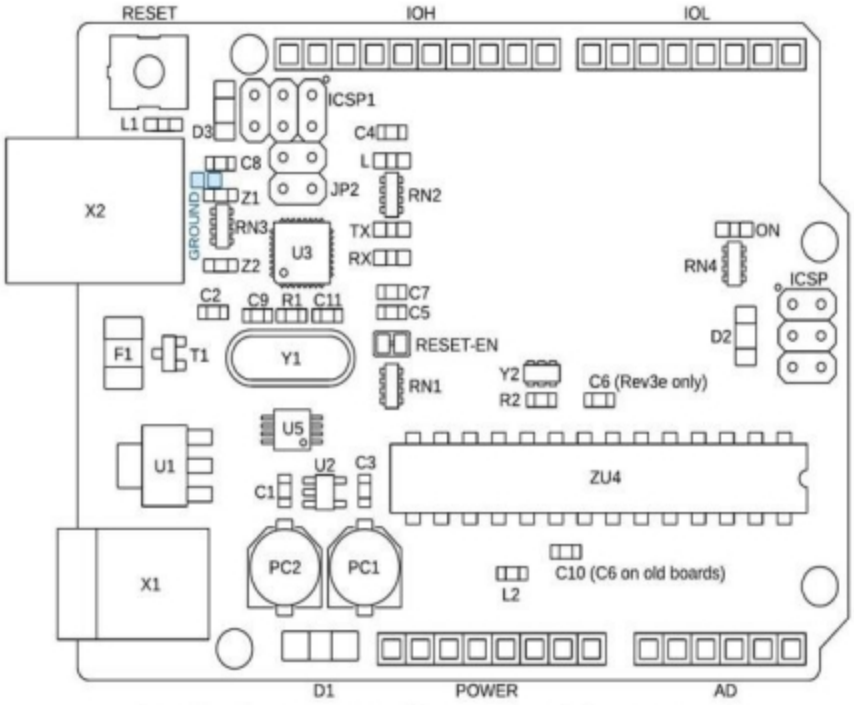


Figure 6.5: Component Positions on the Arduino Uno Board

6.4 Getting an Electronic Copy of the Circuit Diagram

Arduino hardware is open-source, and the circuit diagram and source files for the circuit diagram and board design are freely available to examine and modify within the terms of the Creative Commons license. Find the circuit diagram in PDF format, and the circuit diagram and PCB design file in Eagle format at store.arduino.cc/arduino-uno-rev3 for the Arduino Uno. Look under the DOCUMENTATION tab for these files.

To view the files in Eagle format, the Eagle EDA software package will need to be downloaded from www.autodesk.com/products/eagle/free-download and installed. It is available for Windows, Mac and Linux. Although Eagle is a commercial product, it can run as a limited version for hobbyists that has certain restrictions, but is fine for examining the Arduino Uno schematic and board files.

Chapter 7 • Fault Finding and Measurement

This chapter is dedicated to fault finding on the Arduino Uno. Basic fault finding techniques are covered. An explanation of how to remove a faulty microcontroller from an Arduino Uno and replace it with a new one is included.

In order to test an Arduino Uno board, various measurements need to be made. Test points on the board are shown in order to make voltage measurements to compare to when testing a faulty board. Both DC voltage measurements that can be tested with a multimeter and various AC waveform measurements are covered that can be measured using an oscilloscope.

Examples that capture PWM, UART, TWI and SPI signals on an oscilloscope in this chapter can be used as a reference when setting up an oscilloscope to test these signals on a project.

In this Chapter

- Basic fault finding on the Arduino Uno
- Replacing the Arduino Uno main microcontroller and bootloader firmware
- Making voltage measurements using a multimeter
- Making waveform measurements using an oscilloscope
- PWM, UART, TWI and SPI signal oscilloscope examples

7.1 Basic Fault Finding and Repair

Some basic fault finding on the Arduino Uno has already been covered in Chapter 1. Refer to section 1.5.3 – Basic Testing, for the first things to look for on a faulty Arduino Uno board. Knowing the history of the board can be a big help when fault finding. What happened to cause it to fail? Did it fail while in use, and if so, what was connected to it? If a heavy load was connected to it, a pin on the ATmega328P microcontroller could be burned out, or even the microcontroller itself. If the board was known to be working and then failed the next time it was used, this could point to possible mechanical damage if it was packed away in an unsuitable place, or perhaps damage from static electricity.

Except for the main microcontroller, the crystal for the ATmega16U2 and connectors on an Arduino Uno board, all electronic components are surface mount devices soldered to the top of the board. This makes repair more difficult for the average hobbyist or maker. Some of the bigger devices, such as the 5V regulator, are easier to replace than small devices with fine pitch pin spacing. It is up to each individual to decide whether replacing a part is too difficult or not. Inexperienced users could end up damaging the board more when attempting to do a repair, so discretion is advised.

Replacing the main microcontroller on an Arduino Uno is one of the easier repair jobs to do, which is covered in the next section. In the sections that follow, test points for measuring voltages on an Arduino Uno are shown, which can be used when fault finding in order to determine if voltages at the various points are at the correct levels, or if they indicate a fault.

7.2 Replacing the Main Arduino Uno Microcontroller

Only the Arduino Uno has an easily replaceable microcontroller. The Arduino Uno SMD microcontroller is soldered directly to the top of the board, making it difficult to replace without professional soldering and rework equipment.

If the Arduino Uno main microcontroller is damaged or faulty, replace it with a new device. New microcontrollers are available as blank devices containing no firmware, but some stores offer microcontrollers for the Arduino Uno that have the bootloader firmware already programmed into the Flash memory of these devices. If the bootloader is already loaded to memory, then it is simply a matter of replacing the microcontroller on the

Arduino Uno and nothing more. If the replacement microcontroller is a blank device, then the bootloader will need to be programmed to the device and the internal fuses set correctly.

7.2.1 Replacement Part

Replace the 28 pin DIP packaged microcontroller on the Arduino Uno with a new device that has the part number **ATMEGA328P-PU**. This microcontroller is available from a number of different online stores. For those users who do not have a USB programmer, be sure to get a microcontroller that has the Arduino Uno Optiboot bootloader firmware already loaded to it. These devices should already have the fuses set correctly on the microcontroller, as explained in Chapter 5.

7.2.2 Removing the Old Microcontroller

Chip removal tools are available that can be used to lift a DIP packaged device from its socket, but on the Arduino Uno, the microcontroller package is seated flat on its socket with no space to insert the tool. Instead, use a small screwdriver with a flat tip. Insert the corner of the screwdriver tip between the microcontroller package and its socket at one of the ends of the device. It may be easier to insert the screwdriver slightly off center, rather than trying to insert it exactly in the middle. Instead of trying to lift one end of the microcontroller completely up to remove it, rather lift one end a little bit, then the other end a little bit, until the device comes out of the socket. This is to prevent bending the pins on the opposite side to the side being lifted, although if the microcontroller is verified to be damaged it will not matter if the pins get bent. After lifting one end slightly for the first time, it may be necessary to push that side down a bit in order to get the screwdriver between the microcontroller and its socket on the other end.

7.2.3 Inserting the New Microcontroller

Pins on a new DIP packaged microcontroller are not 90° to the body of the device as Figure 7.1 B on the next page shows, but are usually bent at an angle, as can be seen in Figure 7.1 A, making it more difficult to insert into the socket. To remedy this, place the microcontroller on its side so that the pins on one side are parallel to the workbench or table top (Figure 7.1 C), then bend the pins so that they are closer to 90° to the body by pushing the pins onto the work surface and rolling the body of the device over, as can be seen in Figure 7.1, C and D. The pins do not need to be bent perfectly to 90° to the body, as long as they are closer to 90° than they originally were.

When done, turn the microcontroller over and bend the pins on the other side so that the end result looks like Figure 7.1 B. The microcontroller can now be inserted into the socket a lot easier. Be sure to insert the microcontroller into the socket the correct way around, so that the half circle cut-out at one end of the microcontroller, and dot at pin one, are on the same side as the half circle cut-out on the socket. Pin one and the half circle cut-out on the microcontroller must be on the same end of the board as the main ICSP header, as shown in Figure 7.2.

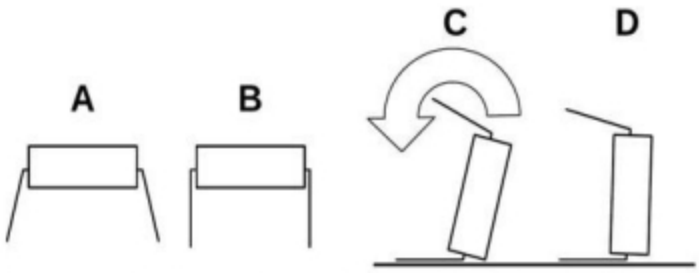


Figure 7.1: Bending the Pins on a New Microcontroller – End View of Device Package

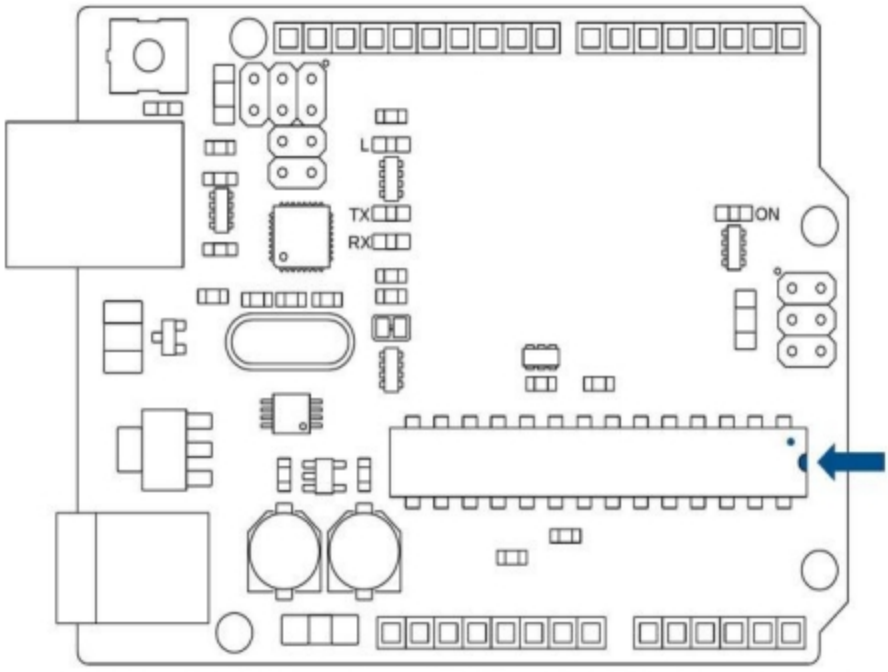


Figure 7.2: Orientation of the ATmega328P Microcontroller on the Arduino Uno

7.2.4 Loading the Bootloader

For those users who have a USB programmer, and who replaced the microcontroller on the Arduino Uno with a blank device that has no bootloader firmware in Flash memory, refer to section 2.8.2 of Chapter 2 for information on how to restore the bootloader using the Arduino IDE. Refer to section 5.4 of Chapter 5 for more information on loading the bootloader to the microcontroller using Microchip Studio. If using Microchip Studio, be sure to also program the fuses in the microcontroller as described in section 5.4.4.

7.3 Voltage Measurements

One of the simpler tests to perform on an Arduino is to measure its voltages at various points on the board in order to determine if faulty power components or power supply are the cause of problems on the board. In the two sections that follow, voltages at certain test points on an Arduino Uno board are shown, so they can be compared to measurements made on a board under test. The first section shows measurements on a board powered from USB, and the second section, a board powered from an external power supply.

Use a multimeter set to voltage when making voltage measurements on an Arduino Uno. If the multimeter is not an autoranging type, be sure to set it on a suitable voltage scale for the voltage being measured. When making measurements, especially on small electronic components, be careful not to short out any pins with the probe tip on the multimeter lead. Leads with very sharp tips are less likely to slip and short pins together.

All voltage measurements are made relative to GND. Insert a jumper wire into one of the GND pins of the socketed headers and then use the alligator clip attachment on the black COM lead of the multimeter to clip onto the other end of the jumper wire. This frees up a hand that would otherwise have to hold the COM lead to a GND point, making it slightly easier to take measurements.

To prevent voltage fluctuations when making DC voltage measurements on the Arduino Uno, it is suggested to load a blank sketch to the Arduino. A blank sketch can be found in the Arduino IDE under File ► Examples ► 01.Basics ► BareMinimum from the top menu. This leaves all hardware in its default state and causes program execution to enter an endless loop so that no hardware is being used, switched or operated. With the blank sketch loaded, the L LED stays on.

7.3.1 Powered from USB

Figure 7.3 shows various test points to take DC voltage measurements at in order to check that power is reaching these areas on the board.

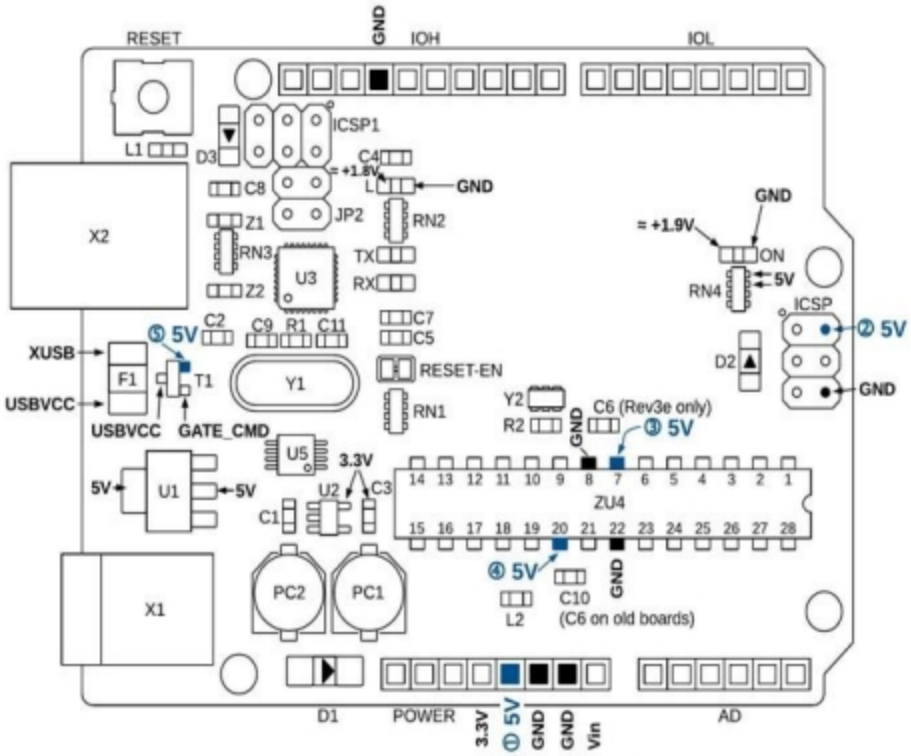


Figure 7.3: Voltage Test Points on an Arduino Uno when USB Powered

7.3.1.1 5V Test Points

First, and most obvious, is to check that 5V is present on the 5V pin of the POWER header socket, as shown in Figure 7.3 at ①. 5V can also easily be measured on the ICSP header as shown at point ② of Figure 7.3. Points ③ and ④ in Figure 7.3 are the two pins that supply the microcontroller with 5V.

7.3.1.2 USB Power and 3.3V Circuit Test Points

The actual USB input voltage can be measured on resettable fuse F1 at the XUSB point that can be seen in Figure 7.3. Be very careful not to short out the probe point of the

multimeter lead to the USB connector casing, which is connected to GND, when measuring the XUSB voltage. On the other end of fuse F1 is USBVCC that will be at a slightly lower voltage than XUSB because of the voltage drop across fuse F1. With the blank sketch loaded to the Arduino Uno and both the L and ON LEDs on, the voltage drop across the fuse will be around 0.03V. As more current is drawn from the Arduino, so the voltage drop across F1 will increase.

USBVCC is fed into transistor T1, which switches it onto the 5V pin of the Arduino Uno at point ⑤, if no external power supply is present and the Arduino is powered from USB alone. The voltage drop across the transistor, when it is on, between USBVCC and 5V is so small that it is insignificant. This 5V is then fed onto the output of the 5V regulator U1, as can be seen in Figure 7.3, where 5V appears on the tab and middle pin of this regulator. Because of the internal protection diode in voltage regulator U1, the 5V voltage appears on the input pin of this regulator where it drops to about 4.5V. Of course the 5V regulator is not used at all when the Arduino Uno is powered from USB, but the 5V being fed back into U1 is simply an effect of this regulator being in the circuit. This feedback voltage of around 4.5V can be measured on the Vin pin of the Arduino.

If 5V reaches the 3.3V regulator U2, then 3.3V should be present on the Arduino Uno 3.3V pin. This voltage can also be measured directly on U2, and on C3, as indicated in Figure 7.3. The output pin of the 3.3V regulator U2 connects directly to C3.

7.3.1.3 Measuring Voltage on the L and ON LEDs

A voltage of around 1.9V to 2V will be measured on the ON LED on the pin indicated in Figure 7.3. 5V is supplied to this LED from two resistors in parallel found in resistor pack RN4. Around 1.8V will be measured on the L LED as indicated in Figure 7.3. Both of these measurements are for a working board when both of these LEDs are on.

7.3.2 Powered from External Power Supply

When an Arduino Uno is powered from external power alone, the input voltage can be measured on diode D1 at the PWRIN point as shown in Figure 7.4 on the next page. VIN is the input voltage on the other side of D1, which is the same as PWRIN minus the voltage drop across D1. The voltage drop across D1 is around 0.7V to 0.8V (actual measurements on two Arduino Uno boards gave approximately 0.76V), which means that if the input voltage PWRIN is 12V, VIN will be around 11.2V to 11.3V. Note that the 0.7V to 0.8V drop across D1 is measured at low load. It can be up to 1.1V with high loads.

When testing the voltages on an externally powered Arduino Uno, first make sure that PWRIN and VIN are present. VIN can be measured on the diode as indicated in Figure 7.4, and also on the Vin pin of the POWER header socket. If PWRIN and VIN are present on the board, make sure that it reaches the 5V regulator U1. If U1 is working properly, the regulated 5V will appear on the output pin of this device, as can be seen in Figure 7.4. If 5V is present on the output pin of the regulator, then it should be present at all of the other 5V measurement points indicated in the figure.

After making sure that 5V is present on the board, check that 3.3V is present on the 3.3V pin of the POWER header socket. 3.3V should be present on U2, which can also be measured on C3, as indicated in Figure 7.4. LED voltages of the L and ON LEDs can also be measured as shown in the figure for these devices when they are both on.

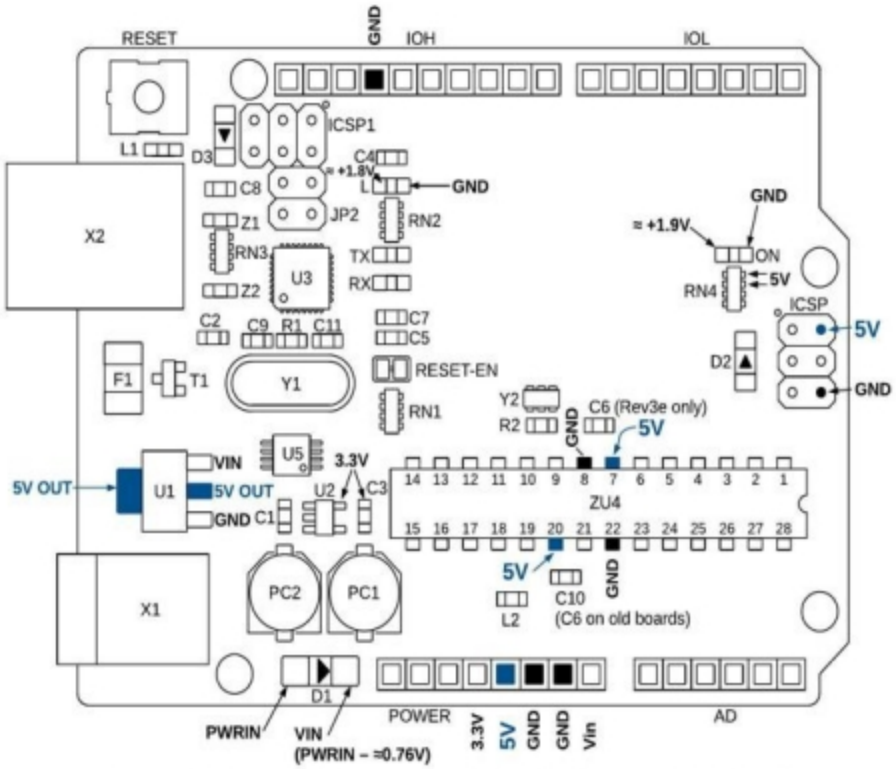


Figure 7.4: Measuring Voltage on an Externally Powered Arduino Uno

7.4 Waveform Patterns and Measurement

Although some multimeters can measure frequency, which can be useful to indicate the presence of an AC signal, one really needs an oscilloscope to properly measure and test an AC waveform pattern or signal. All oscilloscope measurements in this section were made with a PicoScope 2000 series digital PC oscilloscope, model number 2205A. This is a 25MHz bandwidth device that is more likely to be used by hobbyists and makers on a budget. If using a different oscilloscope, make adjustments as necessary to the oscilloscope setups described in the text of this section. Figure 7.5 shows the test points that are used in the sections that follow to test for various AC waveforms.

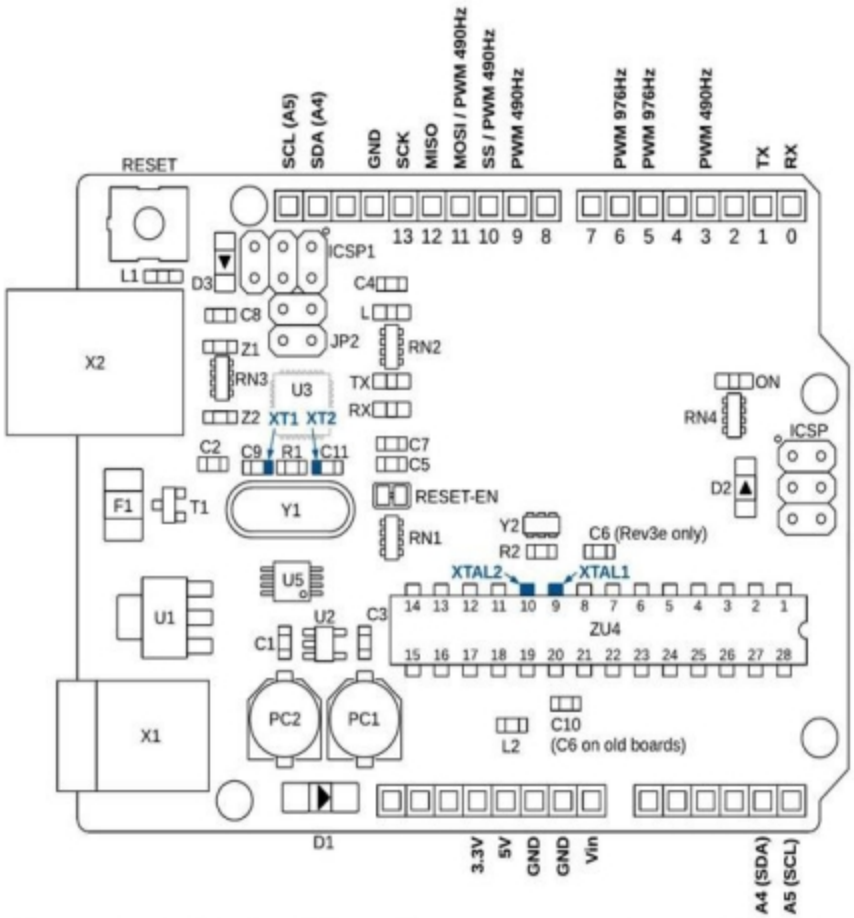


Figure 7.5: Test Points for Making Oscilloscope Measurements on an Arduino Uno

7.4.1 Testing for Presence of Microcontroller Clocks

Testing for the presence of the main clock frequency on a microcontroller can help to determine if the microcontroller is working or not. Both the main microcontroller and USB microcontroller each operate from a 16MHz clock that can be measured on the pins of these microcontrollers, or components attached to these pins. Test for the clock on ZU4, the ATmega328P, on pin 9 or 10, marked XTAL1 and XTAL2 in Figure 7.5.

Figure 7.6 shows the clock frequency measured on the XTAL1 pin of the ATmega328P microcontroller, with a 25MHz bandwidth oscilloscope set to 100 ns/div and the probe set to 10X. As can be seen in the figure, the clock signal is distorted, attenuated and offset from 0V. The reason for the distortion is because the capacitance of the oscilloscope probe interferes with the oscillator circuit. Input capacitance of the probes supplied with the PicoScope 2205A is 14pF to 18pF when set to 10X, and even higher when set to 1X (70pF to 120pF). Some oscillator circuits will actually stop working when an oscilloscope probe is attached to them.

Besides the probe capacitance problem, to capture a proper 16MHz signal, a much higher bandwidth oscilloscope is needed. In this basic test with a lower bandwidth oscilloscope, even though the signal appears distorted, it is still useful to indicate if the clock signal is present. Absence of a clock signal can also indicate that the fuse settings of the ATmega328P are incorrect, see section 5.4.4 for more details.

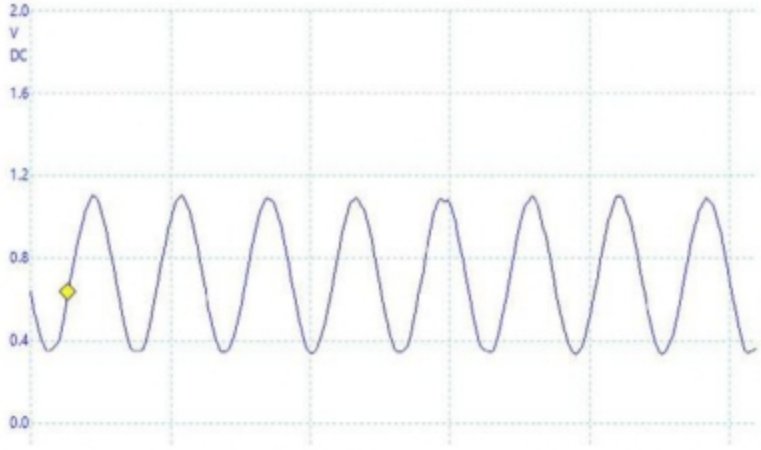


Figure 7.6: 16MHz Clock Frequency on a 25MHz Bandwidth Oscilloscope

Measure the clock frequency of the USB microcontroller, the ATmega16U2 U3, at the XT1 or XT2 test points as shown in Figure 7.5. This clock signal can be measured on C9 or C11, on the ends of these capacitors as indicated in the figure. Expect the signals to look similar to Figure 7.6 because of the probe capacitance. Again, the presence of this signal can help identify if the microcontroller is at least being clocked.

7.4.2 Testing for PWM Waveforms

Figure 7.7 shows two of the PWM signals generated by the Arduino sketch that follows on the next page. Use the sketch to generate a PWM (Pulse Width Modulation) waveform on each of the six PWM pins of the Arduino Uno. As can be seen in the comments of the sketch, each PWM pin is set to generate a square wave signal with a different duty cycle. The top signal of Figure 7.7 is taken from Arduino Uno pin 3 and the bottom signal from pin 11. Use the sketch as a reference when testing the PWM signals on an Arduino Uno project. Refer to section 2.7.4.4 of Chapter 2 for more information on PWM pins.

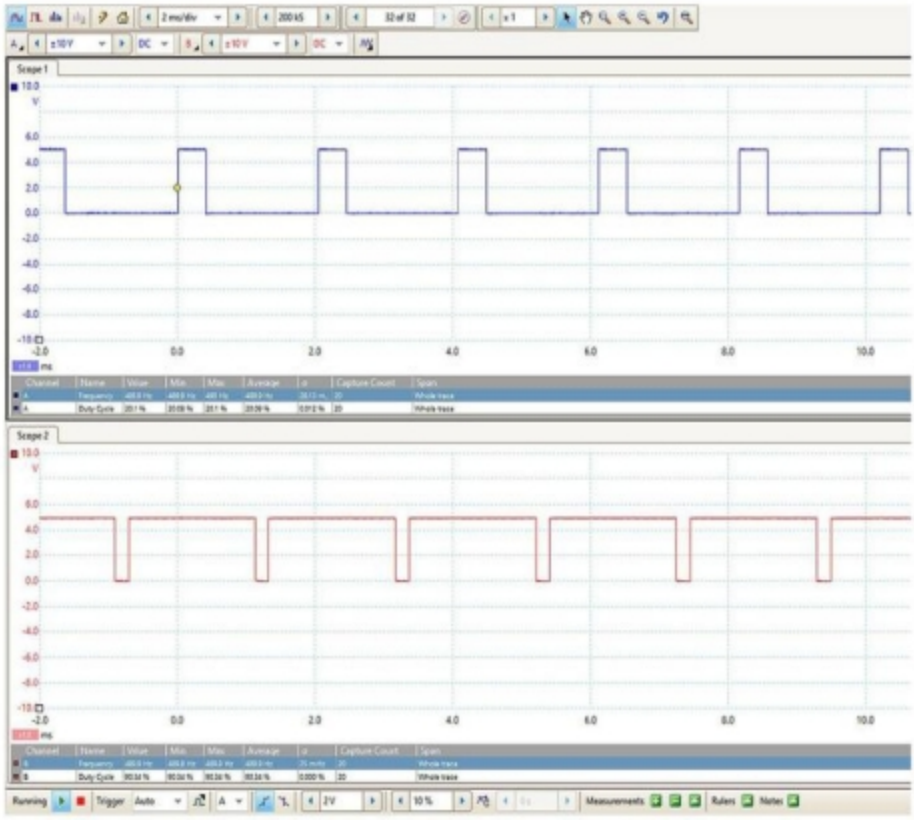


Figure 7.7: PWM Signals from Arduino Uno Pins

```
void setup() {
  analogWrite(3, 51);     // PWM 20% 490Hz
  analogWrite(5, 89);     // PWM 35% 976Hz
  analogWrite(6, 127);    // PWM 50% 976Hz
  analogWrite(9, 166);    // PWM 65% 490Hz
  analogWrite(10, 204);   // PWM 80% 490Hz
  analogWrite(11, 230);   // PWM 90% 490Hz
}

void loop() {
}
```

7.4.3 Testing the UART Output

A square wave can be generated on the UART transmit pin by sending an uppercase U character out of the serial port. The square wave signal can then be measured using an oscilloscope to see if the UART serial port is working. Use the UART test sketch that follows to generate the square wave on pin 1, or TX of an Arduino Uno.

```
void setup() {
  Serial.begin(9600);
}

void loop() {
  Serial.print('U');
}
```

If the UART is working, a square wave with a 50% duty cycle and frequency of approximately 4.8kHz will be seen on the oscilloscope. The frequency will be 4.8kHz for a baud rate setting of 9600.

Some oscilloscopes such as the PicoScope can decode serial data, such as UART data. This can be useful when debugging a project that is sending and receiving serial port data using the UART. In the PicoScope software, the UART data can be decoded by selecting Tools ► Serial Decoding from the top menu. In the dialog box that pops up, click Create and then select UART (RS-232, RS-422, RS-485) from the list that pops up. A new dialog box then opens which allows the oscilloscope channel to be selected as well as the UART settings for the UART under test. Select the appropriate settings and then click the OK button. The oscilloscope software now decodes the UART signal and displays the data. Figure 7.8 shows the UART data from an Arduino Uno running the above sketch.

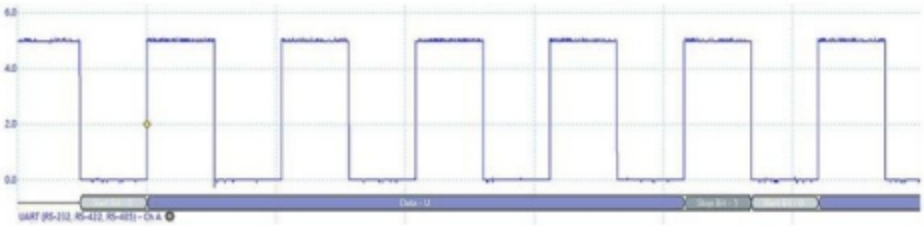


Figure 7.8: Arduino Uno UART TX Signal and Data from a PicoScope

7.4.4 TWI Signals

A PicoScope 2205A can be set up to capture I²C or TWI signals which can aid in fault finding on projects that use TWI devices. Figure 7.9 shows the TWI signals captured while running the sketch that reads the first byte of an EEPROM device from section 3.3.4.2. The TWI clock or SCL is at the top of the figure and the data, or SDA signal, is below it. This figure is split into two parts with the second half placed below the first for a better fit.

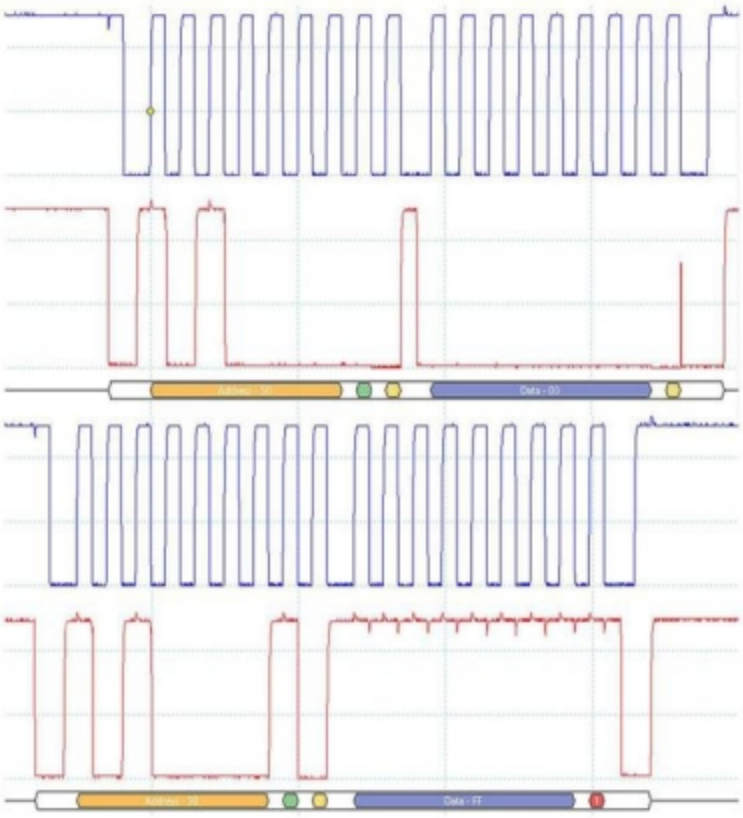


Figure 7.9: Arduino TWI Signals Captured on an Oscilloscope

To capture TWI signals on a PicoScope oscilloscope, start by connecting channel A to the TWI SCL, or clock signal on the circuit under test. Connect channel B of the oscilloscope to the TWI SDA, or data signal of the circuit. For the purposes of this explanation, refer to Figure 3.15 from Chapter 3 for the TWI pinout on an Arduino Uno. It is assumed that a single memory chip is connected to the Arduino, the other TWI device from the figure is not present. Two 4k7 pull-up resistors are used to pull up both the SCL and SDA lines to 5V. This gives much better shaped TWI signals on the oscilloscope. It is also assumed, for the purposes of this explanation, that the sketch from section 3.3.4.2 of Chapter 3 is loaded to the Arduino Uno.

With the oscilloscope attached to the Arduino TWI pins as described above, and the sketch loaded to the Arduino, start the PicoScope software. On the top PicoScope tool menu, enable channel B by changing it from the default Off to ±10V, also change channel A from the default Auto to ±10V, so that both channel A and B are now set to ±10V and DC. If the scope probes are set to 10X, use the drop down menu for channel A and then B on the top toolbar to select x10 for the Probe setting on each channel. Select Tools ▸ Serial Decoding from the top menu and then click Create in the dialog box that pops up. On the menu that pops out, click I2C. In the I2C dialog box, select B for Data and A for Clock. Set the Threshold parameter for each channel to 1V and the Hysteresis for each channel to 20mV. All other parameters in the dialog box can be left at their default, including the Bus Speed setting which should be 100 kbits/s. Click OK in the dialog box when done. Click OK in the Serial Decoding dialog box to close it. Set the time per division setting on the top toolbar to 50 µs/div. Number of samples, found on the top toolbar can be left at the default 16 kS.

By default the oscilloscope should be stopped, if not, click the stop button on the bottom toolbar at the left. On the same toolbar, change the Trigger setting to Single. Now click the green start button on the bottom toolbar at the left. Move the yellow trigger level symbol on the oscilloscope area of the screen up above 0V, and to the left of the screen so that the Threshold is around 2V and Pre-trigger about 10%, as can be seen on the bottom toolbar – hover the mouse cursor over the boxes on the toolbar so that the tool tip pops up to help identify each field. Finally press the RESET button on the Arduino to start the TWI sketch running. If all is set up correctly, the TWI communications will be displayed in the oscilloscope display area of the screen. Click B on the top menu to pop the B channel menu out and then use the Offset field to move the B channel below the A channel signal for easier viewing, by setting it to a negative value. About -30% should be fine.

7.4.5 SPI Signals

To capture SPI signals on a PicoScope, load the following sketch to the Arduino Uno. The sketch simply sends a character out on the MOSI pin continually (digital pin 11).

```
#include <SPI.h>

void setup() {
  SPI.begin();
  SPI.beginTransaction(SPISettings(10000, MSBFIRST, SPI_MODE1));
}

void loop() {
  SPI.transfer('U');
  delay(1);
}
```

Connect channel A of the PicoScope to Arduino pin 13 (SCK) and channel B to Arduino pin 11 (MOSI). The oscilloscope is then set up much the same as it was for TWI signals in the previous section, but with the exception that SPI is selected in the Serial Decoding dialog box, and a higher collection time and higher number of samples is selected in order to get a decent signal on the scope. The PicoScope is set up as follows.

Top Toolbar	
Channel A and B:	±10V, DC, Probe x10 (if probes set to 10X)
Collection Time:	200ms/div
Number of Samples:	2MS
Bottom Toolbar	
Trigger:	Single
Threshold:	2V
Pre-trigger:	0%

Open the SPI dialog box by selecting Tools ► Serial Decoding from the top menu. In the Serial Decoding dialog box, click the Create button and then select SPI from the menu that pops up. Choose the following settings in the SPI dialog box.

SPI Dialog Box	
Data:	B
Clock:	A
Chip Select:	nothing
Threshold (both):	1V

SPI Dialog Box continued...	
Hysteresis (both):	20mV
Clock Edge:	Rising
Chip Select State:	Low (does not matter because this signal is not connected)
Bit Order:	MSB
Display packets in:	ASCII (for Graph and Table)

Click OK in the SPI dialog box after changing the settings. Click OK in the Serial Decoding dialog box to close it. Click the green button on the bottom toolbar at the left to start capturing the SPI signals. When the oscilloscope stops, zoom in using the Horizontal Zoom control on the top toolbar. Offset the data signal from the clock signal by clicking B on the top toolbar and then using the Offset control to change the offset to -30%. To get a decent view of a data character, it is necessary to zoom in to about x5k to x10k. The captured data should then look something like Figure 7.10. Because ASCII was selected in the PicoScope software SPI settings, the data is displayed as an ASCII character – the uppercase 'U' that is sent by the Arduino sketch can be seen in the Data field on the oscilloscope display.

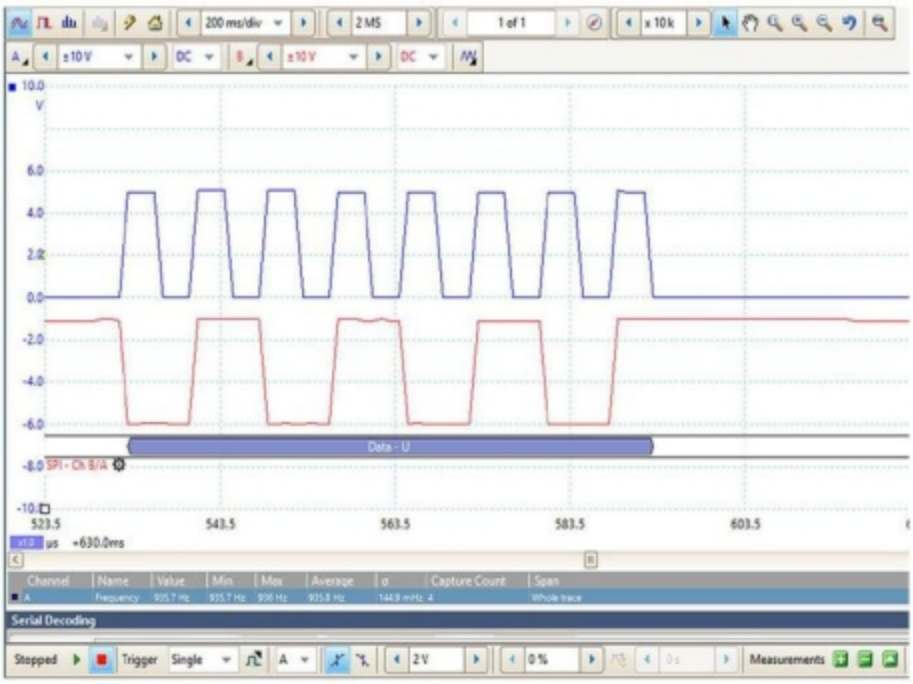


Figure 7.10: Arduino Uno SPI Signals Captured on a PicoScope

Chapter 8 • Mechanical Dimensions and Templates

Dimensions of the Arduino Uno as well as positioning and size of its mounting holes are presented in this chapter.

Dimensions are useful when making an Arduino Uno clone board, making a shield to fit an Arduino Uno, when measuring for placing an Arduino project in an enclosure or when drilling a base plate for mounting.

Rather than including all of the dimensions and sizes on a single drawing, they have been spread over several drawings to prevent clutter and for easier reference.

In this Chapter

- Measurements, tolerance and scale
- Arduino Uno length, width and mass
- Arduino Uno and shield dimensions and spacing

8.1 Measurements, Tolerance and Scale

The Arduino Uno board was designed with the Eagle EDA CAD software package using thousands of an inch as the unit of measurement (a thousandth of an inch is also referred to as a mil). Inches are therefore the "official" measurement system of the board. Dimensions in this chapter are given in inches as well as millimeters. All dimensions are subject to manufacturing tolerances so the actual board size may vary slightly from the measurements given.

Images in this chapter are not to scale. Refer to wspublishing.net for an image of the Arduino Uno that can be printed to scale and used as a template for drilling a base plate when mounting an Arduino. Also find Arduino Uno drill templates at the very back of this book after the index.

8.2 Length, Width and Mass

Table 8.1 and Figure 8.1 show the length and width of the Arduino Uno board as well as the overall length when including the USB connector. As the board was designed using inches, millimeter values have been converted from inches and rounded appropriately.

Table 8.1: Arduino Uno Length and Width

Measurement	Inches	Millimeters
Board Length	2.7in	68.58mm rounded to 68.6mm
Board Width	2.1in	53.34mm rounded to 53.3mm or 53.4mm
Overall Length with USB Connector	2.95in	75mm

The size of the Arduino Uno PCB, or printed circuit board, is given as 2.7 inches long by 2.1 inches wide. This works out to 68.58mm long by 53.34mm wide, which is rounded to 68.6mm by 53.3mm or 53.4mm, depending if rounding the width up or down. At the time of writing, the Arduino Uno REV3 web page gave the dimensions as 68.6mm by 53.4mm, rounding the width up.

Mass of the board is given as 25g (twenty five grams) on the Arduino Uno REV3 web page which works out to 0.88oz (zero point eight eight ounces). Measurements taken on two Arduino Uno boards both gave a mass of 27g or 1oz, and a measurement taken on one Arduino Uno SMD board gave a mass of 23g or 0.8oz. The clear plastic base that is packaged with new Arduino Uno boards has a mass of 11g or 0.4oz. These measurements are of course dependent on the tolerance of the weighing scale used.

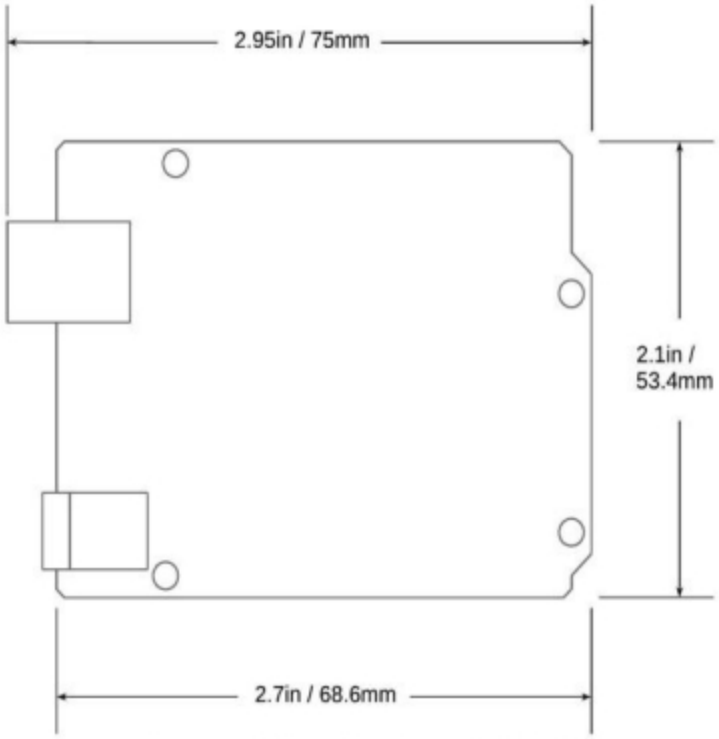


Figure 8.1: Arduino Uno Length and Width

8.3 Mounting Hole Spacing and Size

Arduino Uno boards have four mounting holes near to the corners of the board, but all offset from the corners. Hole size diameter of all mounting holes is 3.2mm or 0.126 inch. Figure 8.2 shows the distances between the mounting holes and edges of the board. Although the hole diameters are in metric, or millimeters, the position of the holes are in imperial or inches – these measurements are displayed in inches and converted to millimeters in the figure.

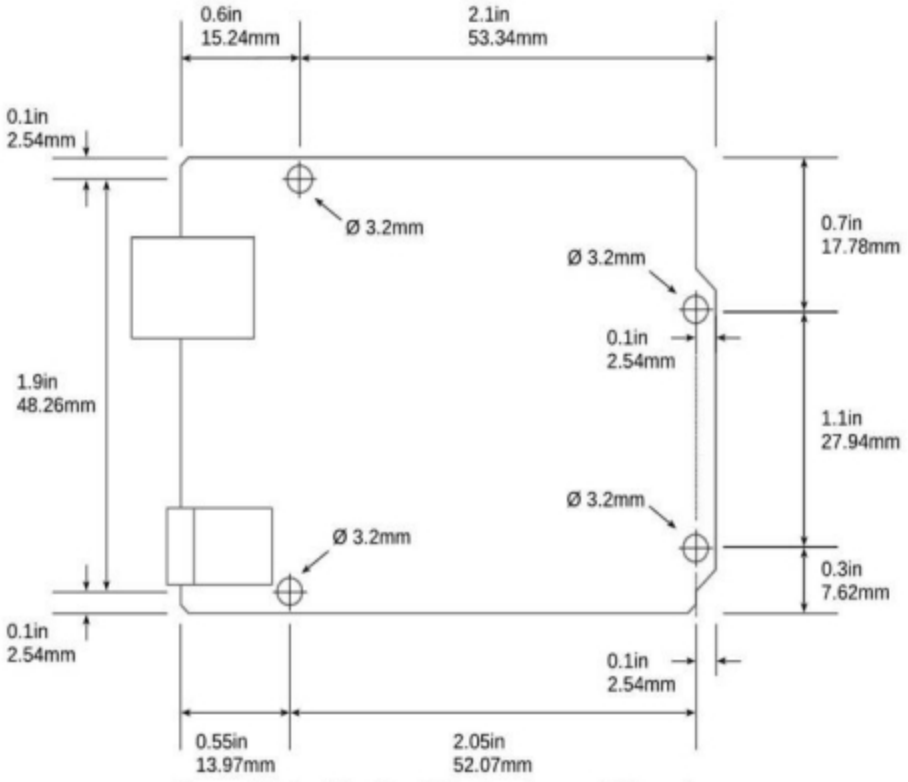


Figure 8.2: Arduino Uno Hole Spacing and Dimensions

Mounting holes were placed on the board in Eagle EDA using a grid of 100mills except for the bottom left hole (near the external power input in Figure 8.2) which was placed using a 50mil grid.

8.4 Shape Dimensions

Shape dimensions refer to the irregular cutout of Arduino Uno boards which have a particular shape cut out at the end of the board. Figure 8.3 shows the dimensions for the cutout shape.

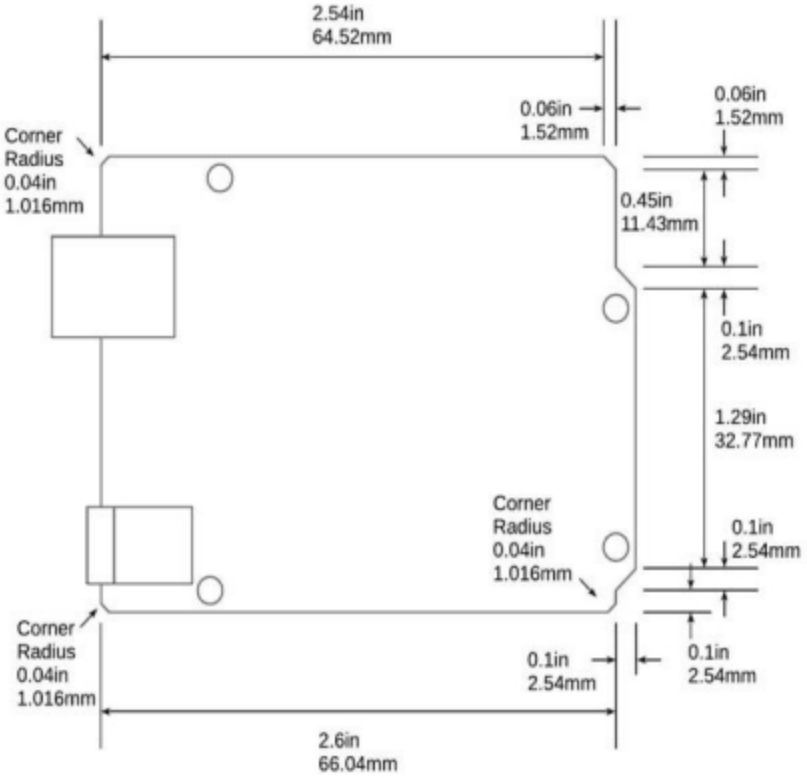


Figure 8.3: Arduino Uno Shape Dimensions

8.5 Header Positions

Figure 8.4 shows the positions for the header sockets at the edges of an Arduino Uno and the ICSP header at the end of the board. These header socket positions are needed when making a shield that fits on top of the Arduino Uno and plugs into the header sockets. Some shields also plug into the ICSP header in addition to plugging into the header sockets, for example the Ethernet shield.

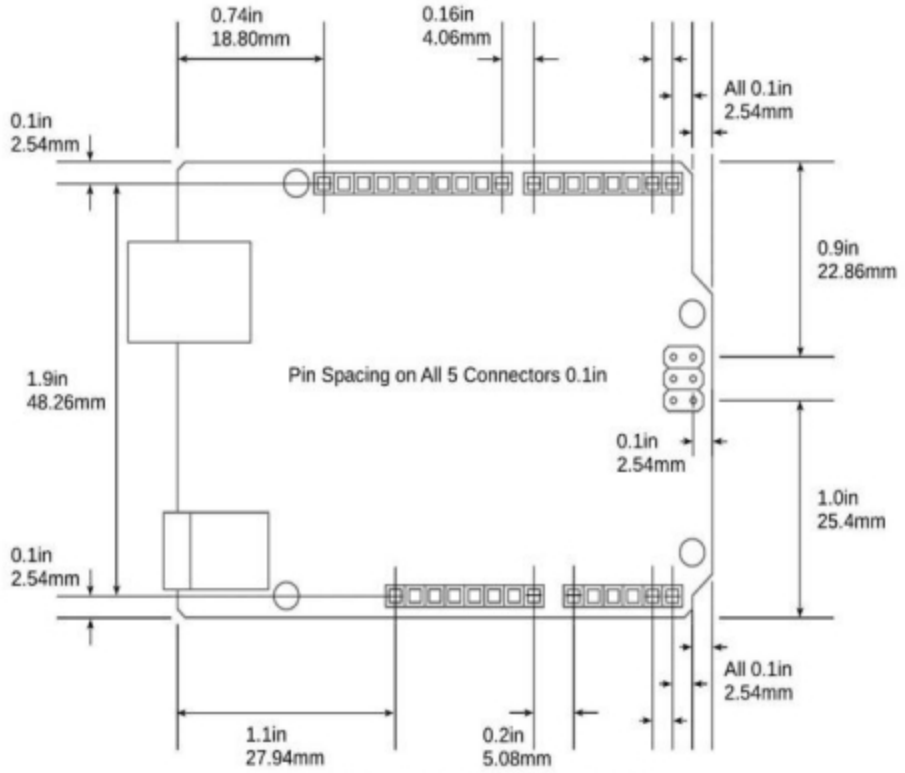


Figure 8.4: Arduino Uno Header Positions

8.6 Shield Reverse Connection Protection

Figure 8.5 shows that three of the header sockets on the Arduino Uno are aligned to a 0.1in or 100mil vertical grid, except for the top left header in the figure that is offset from this grid. A shield that has pins underneath it with the same spacing can not be plugged in the wrong way around because of the offset header that provides reverse connection protection.

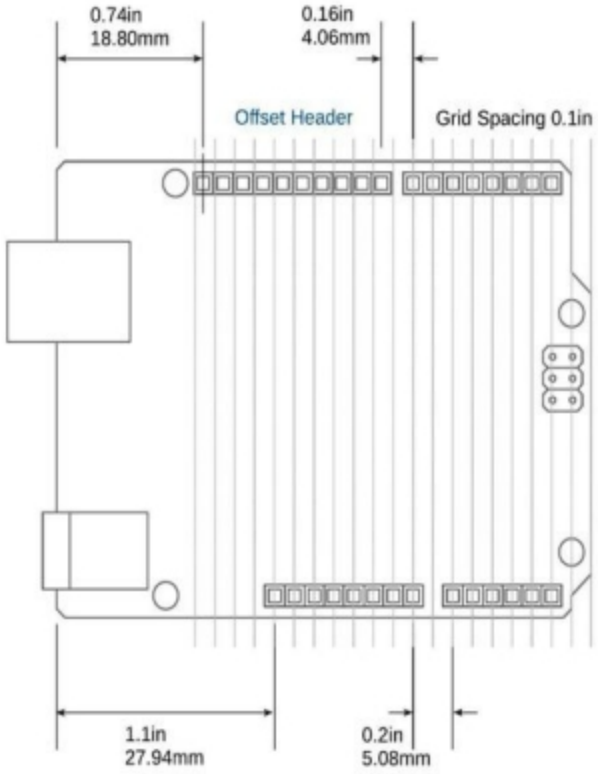


Figure 8.5: Shield Reverse Connection Protection with Offset Header

8.7 Using Strip-board as a Shield

Strip-board, also known by the brand name Veroboard, can be used to make a shield, the only problem being that the offset header that is shown in the previous section does not line up with the holes in the strip-board. If the pins on the offset header are not needed, they can be left unconnected. An alternative is to solder header pins into the strip-board that have long pins, which can then be bent to fit into the offset header. Using a prototype shield is another method that can be used to make a shield, as was discussed in section 1.1.6.2 of Chapter 1.

See startingelectronics.org/projects/arduino-projects/arduino-tiny-stripboard-shield/ for an example of fitting a piece of strip-board to an Arduino Uno using only three of the header sockets on the Arduino. An example project that uses the strip-board shield can be found at startingelectronics.org/projects/arduino-projects/arduino-tiny-temperature-shield/.

8.8 Drill Template and KiCad Template

A drill template, in PDF format, for the mounting holes on an Arduino Uno that can be used when drilling a base plate for mounting an Arduino Uno is available for readers of this book. Be sure to print the file using a 1 to 1 scale so that the printed template is correctly sized. Go to the supporting website at wspublishing.net to find the drill template file. Also find Arduino Uno drill templates at the very back of this book after the index.

Find a KiCad template of an Arduino Uno board at wspublishing.net which can be used to create both a custom Arduino Uno board or shield that will fit the Arduino Uno. KiCad is an open-source EDA CAD package that can be used to draw circuit diagrams and design printed circuit boards. The KiCad software package can be found at kicad.org where it can be downloaded free of charge and unrestricted.

Appendix A • Specifications Quick Reference

The table that follows contains a summary of the Arduino Uno technical specifications with references to the appropriate sections in this book where more information can be found on each parameter.

Parameter	Specification	Notes	Reference
Microcontrollers			
Main Microcontroller	ATmega328P-PU	Arduino Uno	1.1.4.1, 2.1, 5.4, 6.1.2, 7.2
	ATmega328P-MU	Arduino Uno SMD	
USB Bridge Microcontroller	ATmega16U2-MU	ATmega8U2 on boards before rev3	1.1.4.10, 5.3, 6.1.3
Memory			
Flash Memory	31.5K bytes	32K bytes minus 512 bytes for bootloader	2.3, 2.3.1
SRAM	2K bytes	Volatile, data memory	2.3, 2.3.2
EEPROM	1K byte	Non-volatile, data	2.3, 2.3.3
Communications			
UART	1 UART (USART configured as UART)	Universal Asynchronous Receiver Transmitter – serial port, TX and RX pins	2.7.4.5, 2.9.1, 3.3.6, 7.4.3
TWI	1 TWI port, 100kHz default speed	Two Wire Interface, I²C compatible serial bus	2.3.4.2, 2.3.4.4, 2.9.3, 3.1.1, 3.3.4, 7.4.4
SPI	1 SPI port	Serial Peripheral Interface – 4 wire serial bus	2.3.4.1, 2.3.4.2, 2.3.4.3, 2.9.4, 3.1.2, 3.3.5, 7.4.5

Parameter	Specification	Notes	Reference
Operating Frequencies			
ATmega328P Clock Speed	16MHz	Generated from external ceramic resonator	2.5, 7.4.1
ATmega16U2 Clock Speed	16MHz	Generated from external quartz Crystal	2.5, 7.4.1
Voltage Specifications			
Operating Voltage	5V DC	Derived from either USB or external voltage through 5V regulator	1.3.4, 1.5.3.2, 2.4.5, 4.1.1, 6.1.4, 7.3
Input Voltage, USB	5V DC ± 5%	As per USB 2.0 specification	2.4.1, 4.1.2, 4.2.3, 4.3.4, 6.1.3, 7.3.1
Input Voltage, 2.1mm Jack	7 to 12V DC	Center of jack is positive	1.3.4, 2.4.3, 4.1.3, 4.1.4, 4.3.1, 4.3.2, 6.1.4, 7.3.2
3.3V Pin	3.3V DC output		4.2.5, 4.3.3
Current Specifications			
5V Regulator DC Current	Regulator rated at 1.0 A	Current limited by thermal considerations which depend on input voltage and limited heat sinking on board	4.1.1, 4.2.1, 4.3.2
5V USB DC Current	500mA	As per USB 2.0 specification	2.4.1, 4.1.2, 4.3.4
3.3V Pin DC Current	150mA	50mA on pre-Uno boards	2.7.1.3, 3.3.7.3, 4.1.1, 4.3.3
I/O Pin DC Current	20mA	Subject to group limit and maximum limit, see reference	2.7.4.2, 3.3.1.1

Parameter	Specification	Notes	Reference
LEDs			
On LED	Green	Power On Indicator	1.1.4.6, 1.5.3.2, 2.6.1, 4.2.2, 7.3.1.3
L LED	Yellow	User LED	1.1.4.7, 2.6.2, 2.9.2, 7.3.1.3
TX LED	Yellow	Serial TX Indicator	1.1.4.8, 2.6.3
RX LED	Yellow	Serial RX Indicator	1.1.4.9, 2.6.4
Button			
RESET Button	Momentary reset push button	Resets main micro-controller	1.1.4.4, 2.6.5
Pins			
Power Pins	GND, 5V, 3.3V, Vin		2.7.1, 3.3.7, 4.1.5, 4.2.6, 4.3
IOREF Pin	I/O voltage reference	Voltage reference for shield	2.7.2, 3.3.9
RESET Pin	Active low reset input	Resets main microcontroller	2.7.3, 3.3.8
Digital Pins	14 digital I/O pins, 6 of the digital I/O pins are PWM pins marked with ~	6 pins have PWM capabilities, some pins multiplexed with SPI and UART	2.7.4, 3.3.1, 3.3.2, 7.4.2
Analog Pins	6 analog input pins connected to internal ADC	Can be configured as digital I/O pins, two pins multiplexed with TWI	2.7.5, 3.3.3
AREF Pin	Analog reference input, 1V to 5V	Optional external analog input voltage used as ADC reference	2.7.6, 3.3.10

Parameter	Specification	Notes	Reference
Programming Headers			
ICSP	6-pin (3 × 2) pin header	Programming header for main microcontroller	1.1.4.11, 2.8, 2.9.4, 3.4
ICSP1	6-pin (3 × 2) pin header	ICSP header for USB bridge microcontroller	1.1.4.12, 2.8, 2.8.3, 3.5
Power and USB Connectors			
USB	USB Type B	USB 2.0 port	1.1.4.2, 2.4.1, 2.4.2, 4.1.2, 4.2.3
Power Jack	2.1mm Barrel	7V to 12V DC input, center positive	1.1.4.3, 2.4.3, 2.4.4, 4.1.3, 4.1.4
Board Dimensions			
Length	68.6mm	2.7 inches	8.2
Width	53.4mm	2.1 inches	
Mounting Holes			
4 Mounting Holes	3.2mm diameter		1.1.4.13, 8.3
Mass			
Board mass	25g	Measured as 27g	8.2
Firmware			
Optiboot	Main bootloader	Factory loaded to main microcontroller Flash memory	1.2.2, 5.4
DFU Firmware	Device firmware update	Factory loaded to USB bridge microcontroller	5.1, 5.3, 5.3.2
USB Serial Firmware	USB bridge firmware	Factory loaded to USB bridge microcontroller	1.2.1, 5.1, 5.3, 5.3.3

Index

0

0V....69

1

16MHz....66
16MHz clock....170

2

2.1mm barrel connector....63, 129

3

3.3V....39, 66, 69
3.3V pin....69, 119
3.3V regulator....131

5

5V....38, 48, 66, 69
5V pin....69, 118
5V regulator....118, 128, 130

A

A0 to A5....78
AC load....104
AC waveforms....169
active low....119
ADC....78, 81
alternative pin function....92
ANALOG IN....93
analog input pins....78, 110
anti-static....36
anti-static mat....37
anti-static wrist strap....37
Arduino as an ISP....149
Arduino as ISP....83
Arduino IDE....20, 26, 50
Arduino logos....32
Arduino Uno....20
Arduino Uno SMD....20
arduino.cc....34
arduino.org....34
AREF....80
AREF pin....120
AT24C16C....111
ATmega16U2....24, 25, 35, 41, 69, 137
ATmega16U2 firmware....134
ATmega16U2 fuses....141
ATmega328 datasheet....92
ATmega328P....22, 57, 90
ATmega328P fuses....144
ATmega328P-MU....56
ATmega328P-PU....56
ATMEGA328P-PU....163
ATmega8U2....41, 137
Atmel....57
Atmel Studio....25, 135
Atmel-ICE....25, 81, 135, 148
automatic switch....118, 131
AVR....56, 57
AVR Dragon....148

B

bandwidth....169
barrel connector....63, 129
battery....64, 119
baud rate....46
blank sketch....165
block diagram....152
bootloader....35, 43, 82, 142
bootloader size....58
breadboard....30, 50

Burn Bootloader....82
buzzer....103

C

C language....25, 148
ceramic resonator....66
circuit diagram....152
clock frequency....170
clone....56
clone boards....32
COM port....42
compatible boards....33, 56
component list....156
component positions....159
const....72
Creative Commons....32
crystal....66
CS....114
current limiting resistor....95
current per pin....99
current sink limit....101
current sinking....98
current source limits....99
current sourcing....97

D

data retention....58
datasheets....125
DC....23
DC voltage measurements....166
debugging....148
debugWIRE....148
Device Manager....42
DFU....26, 134
DFU bootloader firmware....139
DFU firmware....137
DFU programming....134
Diecimila....40
digital pins....70
dimensions....178
documentation....51
drill template....184
drivers....42
Duemilanove....40
duty cycle....75

E

Eagle EDA....160
EDA....31, 51
EEPROM....59, 111
EEPROM chips....113
EEPROM memory....147
ESD....36
Ethernet shield....114
external memory....59
external power....23, 39, 63, 128, 129
external power supply....69

F

fault finding....41, 48, 162
FCC compliance....34
firmware....24, 35
Flash memory....57
Flash memory size....58
Flash wear....58
FLIP....134
floating pin....78
forum....51
frequency....66
Fritzing....51
FT232RL....35, 69
FTDI....35
fuse settings....25

G

Genuino....34

GND....69
GND pins....118
GPL....32

H

handling precautions....39
hardware revision....40
header socket positions....182
header sockets....68
Header sockets....23
heap....58
History....40

I

I/O....61
I/O pins....93
I2C....61
I²C....61, 111
ICSP....81
ICSP header....24, 25, 84, 88, 121
ICSP header pins....122
ICSP SPI pins....84
ICSP1....122
IIC....61
input pins....73
interfacing....39, 61, 93, 105
internal pull-up....73
internal pull-up resistors....108
internal reference....81
IOREF....41, 69
IOREF pin....120
ISP....121, 136

J

JP2 header....123

K

KiCad....31
KiCad template....184

L

L LED....24, 67, 71, 83
LED....66, 94
LED forward voltage....94
LED_BUILTIN....67, 71, 93
length....178
level shifters....59
LGPL....32
library reference....50
license....32
loading a sketch....43
logo....32

M

main microcontroller....56
main microcontroller circuit....153
maximum current per pin....73
memory....57
Microchip....56, 57
Microchip Studio....25, 135
mil....178
MISO....114
MOSFET....158
MOSI....114
mounting holes....26, 180
multimeter....165
multiplexed....92
multiplexed pins....79

N

NG....40
non-volatile memory....57

O

official Arduino board....33
offset header....183
ON LED....24, 43, 48, 67, 130
op-amp....68, 131

open-source....32
open-source projects....51
operating frequency....66
operating voltage....66, 128
Optiboot bootloader....143
order codes....56
oscilloscope....169
output pin....70
output pins....93
overload....132
overrating....73

P

PCB....31
PDIP....56
PicoScope....169
pin....23
pin current rating....72
pin functions....88
pin headers....68
pin mapping....90
pin numbers....92
pins....68
PN2222....104
ports....92
potentiometer....78
power....38, 61
POWER....65
POWER header socket....131
power jack....129
power pins....69, 118
power supply....63
power supply circuit....129, 155
precautions....36, 39
Processing....52
programming....26
programming headers....81
projects....21
protection circuits....132
protection diode....69
Proto Shield Rev3....29
prototype shields....29
pull-down resistor....106
pull-up resistor....107
pull-up resistors....73
PWM....109
PWM frequency....76
PWM pins....70, 74, 109
PWM signals....171

Q

QFN....56
quartz crystal....66

R

R3....41
reference voltage....78, 80
References....49
relay....104
repair....162
replacement microcontroller....163
RESET....43
reset button....23, 66, 68
reset pin....70, 119
RESET-EN....148
resettable fuse....62, 130, 132
resistor....94
resources....17, 49
REV3....41
reverse connection protection....29, 183
reverse polarity protection....132
revisions....40
RISC....57
RS-232....117
RTC....61

RX....77
RX LED....24, 68

S

safety-critical applications....34
SCK....114
SCL....111
SD card....114
SD card adapters....114
SD cards....59
SD library....116
SD mode....60
SDA....111
serial bus interfaces....28
serial data....172
Serial Monitor window....46
serial port....77, 116
Serial Port Demonstration....46
series resistor....40, 94
shared pins....83
shields....28, 68, 182
short circuit protection....132
silkscreen....41
sketches....17, 26
SMD....20
socket....56
software serial library....117
solder pad jumpers....159
solder pads....123
SPI....61
SPI bus pins....114
SPI library....116
SPI mode....60
SPI pins....88
SPI signals....175
SRAM....58
SS....114
stack....58
static electricity....36
static sensitive....37
static shielding....36
strip-board....30, 184
supporting website....17
switch....106

T

technical specifications....185
template file....184
thermal shutdown....132
through-hole....21
TQFP....56
trademark....34
trademarks....32
transistor....103, 131
TTL....117
tutorial....50
TWI....61
TWI bus pins....111
TWI pins....79, 83, 88
TWI signals....173
TX....77
TX LED....24, 68

U

UART....47, 77, 116, 172
UART test....172
UGND....130
unity gain op-amp....68
unused pin....70
USART....116
USB 2.0....62
USB bridge....24
USB bridge chip....77
USB cable....23, 62
USB connector....23

USB microcontroller circuit.....154
USB power.....128
USB programmer.....148
USB programming device.....135
USB to serial bridge.....137
USB to serial bridge firmware.....139
USB to UART bridge.....35

V

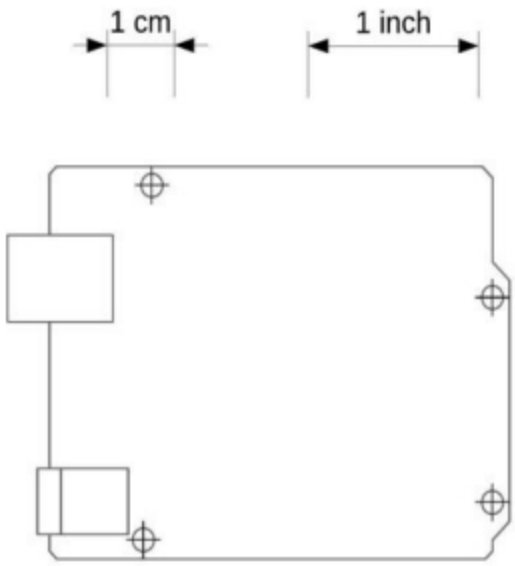
Veroboard.....30, 184
Vin.....64, 69
Vin pin.....119, 129
virtual COM port.....35
volatile memory.....58
voltage.....38
voltage measurements.....165

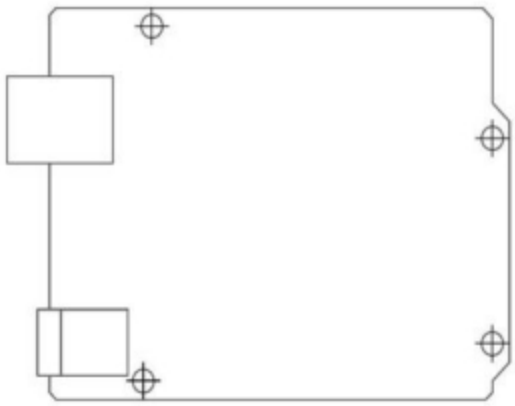
W

warranty.....34
weak pull-up resistor.....109
width.....178
Wire library.....61, 113
Wiring.....21, 52

#

#define.....72





Arduino Uno drill templates referenced in section 8.1 and 8.8 of Chapter 8.

Check the dimensions before using to ensure that the images were not distorted during printing.

elektorbooks

A Reference and User Guide for the Arduino Uno Hardware and Firmware

ULTIMATE Arduino Uno Hardware Manual

A manual providing up-to-date hardware information for the popular Arduino Uno, the easy to use open-source electronics platform used by hobbyists, makers, hackers, experimenters, educators and professionals.

Get all the information that you need on the hardware and firmware found on Arduino Uno boards in this handy reference and user guide.

- Ideal for the workbench or desktop.
- Contains all of the Arduino Uno hardware information in one place
- Covers Arduino / Genuino Uno revision 3 and earlier boards
- Easily find hardware technical specifications with explanations
- Pin reference chapter with interfacing examples
- Diagrams and illustrations for easy reference to alternate pin functions and hardware connections
- Learn to back up and restore firmware on the board, or load new firmware
- Basic fault finding and repair procedures for Arduino Uno boards
- Power supply circuits simplified and explained
- Mechanical dimensions split into five easy to reference diagrams
- Contains circuit diagrams, parts list and board layout reference to easily locate components

Warwick A. Smith is an electronics engineer and embedded programmer from South Africa with experience in industrial, commercial and aerospace related industries. Warwick has a wide interest in various fields of technology, including embedded systems hardware, software, and information technology. His writing style has been described as "clear and concise" as well as "conversational and friendly".

Elektor International Media BV
www.elektor.com

ISBN 978-3-89576-434-9
9 783895 764349

elektor
design › share › sell